
Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

Rui Jorge
Gregório Deyllot

Diplomacy – Base de Dados de Movimentos para Controlar Províncias

Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

Rui Jorge Gregório Deyllot

Diplomacy – Base de Dados de Movimentos para Controlar Províncias

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Pedro Lopes da Silva Mariano, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Dr. João Pedro Neto, Professor Auxiliar da Faculdade de Ciências da Universidade de Lisboa

Dedico este trabalho aos meus tios pelo incansável apoio.

o júri

presidente

Prof. Dr. Joaquim Arnaldo Carvalho Martins
Universidade de Aveiro

Prof. Dr. Francisco Manuel Gonçalves Coelho
Departamento de Informática da Universidade de Évora

Prof. Dr. Pedro Lopes da Silva Mariano
Universidade de Aveiro

Prof. Dr. João Pedro Guerreiro Neto
Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa

agradecimentos

Neste momento quero expressar a minha gratidão a todas as pessoas que contribuíram directa e indirectamente para que o presente trabalho pudesse ser uma realidade.

Primeiramente agradeço ao Professor Doutor Pedro Lopes da Silva Mariano, orientador interno deste trabalho, pela sua dedicação, orientação e apoio que me prestou. Agradeço também ao Professor Doutor João Pedro Neto, orientador externo deste trabalho, pela sua orientação e interesse mostrados apesar da distância física existente.

Expresso também a minha profunda gratidão aos meus tios, Carla e António, que me acolheram em sua casa durante os últimos cinco anos e cujo apoio e companheirismo foram indispensáveis durante a minha progressão académica. Aos meus pais pelo esforço material que realizaram durante estes anos e a toda a restante família pelo apoio e motivação que sempre me deram.

Por último, não posso deixar de agradecer a todos os amigos que adquiri, dentro e fora do ambiente académico, pelos muitos momentos de alegria que me proporcionaram.

A todos um “muito obrigado”.

palavras-chave

diplomacy, inteligência artificial, teoria de decisão, teoria de jogos, minimax, enumeração de grafos.

resumo

A literatura sobre modelos e ferramentas para a construção de agentes para jogarem jogos a 2 jogadores é relativamente extensa. Algoritmos de pesquisa, bases de dados de aberturas ou com jogadas finais, ou heurísticas especialmente desenvolvidas são algumas das técnicas mais utilizadas.

O Diplomacy é um jogo para 2 ou mais jogadores, que decorre num mapa que retrata o estado da Europa no início do século 20. O mapa encontra-se dividido em províncias. Certas províncias são denominadas centros de produção e o objectivo do jogo consiste no controlo de mais de metade dos centros de produção. Cada jogador controla um certo número de unidades.

O factor de ramificação do jogo é elevado. Há cerca de 4 mil biliões de aberturas possíveis. No entanto há implementações que calculam a jogada que cada uma das suas unidades deve fazer. Para cada unidade é efectuada uma pesquisa local de modo a encontrar qual a acção que deve efectuar.

Este trabalho apresenta uma possível abordagem a este problema que passa pela construção de uma base de dados de movimentos tendo em conta o controlo de uma província dentro de um determinado horizonte de turnos. A construção da base de dados é feita à custa de mecanismos de pesquisa de soluções e enumeração de mapas possíveis.

keywords

diplomacy, inteligência artificial, teoria de decisão, teoria de jogos, minimax, enumeração de grafos.

abstract

The literacy about models and tools for the construction of agents capable of playing 2 player games is relatively extense. Search algorithms, data bases of opening or final moves, or specially developed heuristics are some of the most used techniques.
Diplomacy is a game for 2 or more players in a map depicting the state of Europe in the beginning of the 20th century. The map is divided in provinces. Certain provinces are named supply centers and the goal of the game is to control more than half of the supply centers. Each player controls a certain number of units. The game's branching factor is very high. There are about 4 thousand billion possible openings. However, there are programs that calculate the move that each one of the units should make.
This work presents one possible approach to this problem by the construction of a movements database considering the control of a province within a certain number of turns. The construction of the database is made through the use of solution search and map enumeration mechanisms.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Diagramas

Lista de Abreviaturas

API *Application Programming Interface*

DAIDE *Diplomacy Artificial Intelligence Development Environment*

GUI *Graphical User Interface*

IDE *Integrated Development Environment*

1

Introdução

1.1 Motivação & Objectivos

O *Diplomacy* é um jogo de tabuleiro para 7 jogadores que se desenrola num mapa. Desde a sua criação o *Diplomacy* sempre colocou dificuldades ao desenvolvimento de agentes de inteligência artificial capazes de se comportarem de forma realista e eficiente devido ao alto teor de interactividade humana e capacidade de antecipação que o jogo exige dos seus jogadores. O facto de qualquer jogador precisar de prever o melhor possível as acções dos seus oponentes, coloca desafios ao desenvolvimento do software.

Desde a criação da primeira versão electrónica deste jogo, em 1984, que o desafio de desenvolver um agente de inteligência artificial tem aliciado programadores e não só. No início do século actual assistiu-se ao aparecimento de novos projectos relativos ao jogo.

Tem-se assistido ao desenvolvimento das áreas de estudo relativas à inteligência artificial, como por exemplo as áreas de processamento estatístico, teorias de jogos e algoritmos de pesquisa. Este desenvolvimento aumenta o interesse para a aplicação destes conhecimentos nos domínios da robótica e do software de entretenimento devido ao papel fundamental que a inteligência artificial tem nestes domínios.

Este trabalho destina-se sobretudo a desenvolvedores de agentes de inteligência artificial, para o jogo *Diplomacy*, pela possibilidade de integração do software desenvolvido como parte de um agente. No entanto, o trabalho poderá também ser útil a qualquer jogador que queira ter um mecanismo informatizado para planear as suas jogadas.

O objectivo principal do trabalho é construir uma base de dados de movimentos. Para tal é necessário implementar uma solução para o problema de planeamento e previsão de movimentos com recurso ao princípio algorítmico *Min-Max* (também conhecido por *Minimax*). Cada registo na base de dados representará um subestado possível do jogo.

Como objectivo secundário, encontra-se a criação de um enumerador de situações de jogo possíveis para efeitos de inicialização da base de dados. Este enumerador assentará em princípios de enumeração e isomorfismo de grafos, e deverá ser capaz de criar situações de

jogo, passíveis de serem resolvidas pelo sistema de planeamento, para serem armazenadas na base de dados. Será importante garantir que o enumerador não criará situações equivalentes para não criar redundância desnecessária na base de dados.

1.2 Pressupostos

O trabalho desenvolvido nesta tese, e estudado neste documento, apresenta alguns pressupostos recomendados para uma melhor compreensão dos mesmos. De seguida se enunciam esses pressupostos:

1 Recomenda-se a posse de bons conhecimentos sobre *Programação Orientada por Objectos* de forma a compreender com mais facilidade a estrutura e estratégia de implementação do software desenvolvido. Visto os objectivos deste trabalho serem o estudo do princípio *Min-Max* e de técnicas de *Enumeração de Grafos*, não será feito nenhum estudo exaustivo do *Paradigma de Programação Orientada por Objectos*, sendo apenas apresentadas noções básicas sobre objectos, classes e algumas noções sobre as ferramentas *JUnit* e *JMock* para programação em *Java*.

2 Recomenda-se a posse de alguns conhecimentos na área de *Algoritmos e Complexidade*, nomeadamente em termos de reconhecimento de *Ordens de Complexidade* e sua relevância nesta área. Estes conhecimentos poderão proporcionar uma mais fácil compreensão da aplicação do algoritmo recursivo *Min-Max* neste problema.

3 Apesar de poder ser útil ao leitor, a posse de conhecimentos nas áreas matemáticas de *Teoria de Jogos* e *Teoria de Decisão* não é estritamente necessária para a compreensão deste trabalho visto ser fornecida uma breve introdução a esses temas aquando da apresentação ao leitor do *Princípio de Decisão Min-Max*;

4 Por fim, recomenda-se que o leitor possua conhecimentos ao nível das áreas de Matemática Discreta, Recursividade e Estruturas de Dados, mais concretamente sobre Grafos e Árvores, de forma a mais facilmente compreender o uso de um grafo como estrutura de suporte à representação de um mapa, e a representação de um processo recursivo em forma de árvore.

1.3 O Jogo Diplomacy

1.3.1 Descrição do jogo e sua história.

Diplomacy é um jogo de estratégia de guerra cujo cenário se situa na Europa do início do século XX, pouco antes da Primeira Guerra Mundial. O jogo foi originalmente criado por *Allan B. Calhamer* em 1954 e publicado pela primeira vez em 1959, sendo nessa altura um jogo de tabuleiro com capacidade para um máximo de sete jogadores em simultâneo [1]. Desde então o jogo expandiu-se rapidamente com múltiplas edições em tabuleiro e torneios desde 1970. Para além de ser possível jogar o *Diplomacy* com um grupo de amigos, há quem jogue por correio normal ou utilizando uma ferramenta informática. Neste último caso, os exemplos variam desde a utilização de correio electrónico até aplicações cliente/servidor. Com esta solução, o servidor é responsável por manter o estado do tabuleiro e processar as jogadas. Os clientes podem estar associados a jogadores humanos ou a agentes artificiais.

A primeira versão electrónica do *Diplomacy* data de 1984 publicada pela empresa *Avalon Hill* [4]. Recentemente a empresa *Paradox Interactive* [3], conhecida publicadora de software na área de entretenimento, publicou uma nova versão electrónica do *Diplomacy* em 2005.

1.3.2 Características e Regras do jogo.

1.3.2.1 Breve descrição:

O *Diplomacy* distingue-se de muitos outros jogos de estratégia de guerra em tabuleiro por apresentar a ausência, por completo, de factores geradores de aleatoriedade, como por exemplo a inclusão de dados. Tal como o próprio nome indica, o *Diplomacy* baseia-se na arte de conquistar os seus objectivos através de interacções diplomáticas como o estabelecimento e quebra de alianças, entre outras. Estas interacções simulam o clima de negociação e intriga existente na época histórica retratada no *Diplomacy*.

1.3.2.2 O mapa:

O mapa do jogo contém toda a Europa e partes do Norte de África e Médio Oriente. O mapa encontra-se dividido em regiões ou províncias, mais exactamente cinquenta e seis regiões terrestres e dezanove regiões marítimas. As regiões terrestres dividem-se ainda em regiões interiores e regiões costeiras. Das cinquenta e seis regiões terrestres, trinta e quatro contêm centros logísticos denominados *"Supply Centres"*. O objectivo dos jogadores passa por controlar dezoito *Supply Centres*, ou seja, mais de 50% dos *Supply Centres* presentes no mapa para ser considerado o vencedor. Cada jogador inicia o jogo já com alguns *Supply Centres* na sua posse, denominados *"Home Supply Centres"*.

1.3.2.3 O exército:

O exército de cada jogador é constituído por unidades terrestres e marítimas sendo que todas as unidades têm o mesmo peso entre si. Isto implica que numa batalha ganha quem tiver vantagem numérica. Cada região só pode conter uma unidade dentro de si em simultâneo sendo que as regiões terrestres costeiras podem ser ocupadas tanto por unidades terrestres como marítimas. Cada jogador só pode ter uma unidade por cada *Supply Centre* que controla.

1.3.2.4 Fases do jogo:

O jogo inicia-se no ano de 1901 e procede por fases de jogo sendo cada ano composto pelas seguintes cinco fases:

- Negociação de Primavera;
- Movimentação de Primavera;
- Negociação de Outono;
- Movimentação de Outono;
- Fase de Fim-de-Ano.

Durante as fases de negociação, os jogadores interagem entre si, de forma verbal ou de outra forma pré-determinada, de forma a forjarem alianças públicas e/ou secretas entre si. Após cada fase de negociação segue-se uma fase de movimentação na qual os jogadores escrevem os movimentos/instruções que cada uma das suas unidades vai executar. Quando todos os jogadores tiverem terminado a escrita dos seus movimentos, os mesmos são executados em simultâneo, sendo analisadas todas as situações de batalha e procedendo-se a instruções de retirada e ajustamentos resultantes das várias batalhas. Por fim, na fase de

fim-de-ano é feita a actualização da posse dos *Supply Centres*. Um *Supply Centre* que contenha uma unidade passa a ser controlado pelo jogador a que pertence a unidade. *Supply Centres* desocupados não mudam de jogador. Os jogadores que possuam mais unidades do que *Supply Centres* são forçados a retirar as unidades em excesso enquanto os jogadores com menos unidades do que Supply Centres têm a possibilidade de construírem mais uma unidade por cada *Supply Centre* a mais. As unidades construídas são posicionadas uma em cada *Home Supply Centre* não ocupado. Poderá dar-se o caso de um jogador não poder criar todas as unidades pretendidas por não ter os seus *Home Supply Centres* desocupados. Qualquer jogador que não possua nenhum *Supply Centre* nesta fase é retirado do jogo. Se houver um jogador com dezoito ou mais *Supply Centres* o mesmo é declarado o vencedor.

1.3.2.5 Movimentos e batalhas:

As unidades deslocam-se e enfrentam-se no campo de batalha recorrendo aos seguintes cinco movimentos:

- *Move;*
- *Hold;*
- *Support Move;*
- *Support Hold;*
- *Convoy;*

As unidades utilizam o movimento "*Move*" para se deslocarem ou atacarem uma região adjacente e utilizam o movimento "*Hold*" para permanecerem ou defenderem a região que ocupam. O movimento "*Convoy*" só pode ser executado por unidades marítimas servindo para transportar unidades terrestres entre duas margens de uma massa de água. Como todas as unidades têm o mesmo peso entre si, a superioridade numérica necessária para vencer uma batalha é conseguida à custa de movimentos de suporte. O movimento "*Support Move*" permite a uma unidade oferecer suporte atacante a um amigo ou aliado que esteja a atacar uma região adjacente a ambas as unidades. Por outro lado, o movimento "*Support Hold*" permite que uma unidade ofereça suporte defensivo a um amigo ou aliado que esteja a defender uma região adjacente à sua. Para que uma unidade consiga conquistar uma região, tem que obter, com a ajuda de suportes se necessário, um peso maior que o peso obtido pela estrutura defensiva inimiga, caso contrário ambas as unidades permanecem nos seus lugares por se cancelarem entre si. Por fim, uma unidade pode impedir uma unidade

inimiga de oferecer suporte se a atacar enquanto esta efectua o suporte. Nesse caso, essa unidade não acrescentará peso à unidade que está a suportar.

1.3.3 Inteligência Artificial no *Diplomacy* – Estado da Arte.

A grande dificuldade de construção de um agente de inteligência artificial, que seja capaz de jogar o jogo de forma coerente e eficiente, prende-se com dois factores principais.
O primeiro é, obviamente, o facto de o *Diplomacy* envolver um grande nível de interacção humana entre os vários jogadores. A constante necessidade de negociação, seja ela verbal, textual ou outra, apresenta um grande desafio à elaboração de software capaz de participar nessa interacção de forma inteligente. Para tal seria necessário desenvolver um protocolo de conversação que poderia tornar-se algo restritivo para os jogadores humanos.
O segundo factor prende-se com o facto dos movimentos das várias unidades serem todos executados em simultâneo. Devido a esta característica do jogo, não existe a noção de turnos individuais entre os jogadores, o que dificulta, a qualquer agente de inteligência artificial, o processo de decisão dos seus movimentos com base no estado actual do mapa visto que todas as unidades dos outros jogadores se irão movimentar juntamente com as suas. Esta característica do jogo obriga a que seja necessário encontrar uma forma não só de decidir quais os movimentos a executar, mas também uma forma de prever o que os outros jogadores irão fazer, à semelhança daquilo que o tão conhecido supercomputador *DeepBlue* [5] faz para um jogo de xadrez. É para este segundo factor que se pretende apresentar uma solução eficaz neste trabalho.
Em Janeiro de 2002 um grupo de programadores criou o projecto *DAIDE* [6] (Ambiente de Desenvolvimento de Inteligência Artificial para *Diplomacy*) que consiste numa plataforma de desenvolvimento de agentes de inteligência artificial para o *Diplomacy* com os seguintes recursos:

- Um modelo e um protocolo de comunicações;
- Uma linguagem na qual se podem expressar negociações e instruções;
- Um “árbitro” para o jogo;
- Bibliotecas de funções para facilitar o desenvolvimento de software;
- Vários agentes de inteligência artificial muito básicos, denominados “*Bots*”, contra os quais se pode testar um *bot* por si criado.

Neste momento, o ambiente de desenvolvimento já está criado, e a sua equipa encontra-se numa fase de criação de *bots* mais inteligentes e eficientes para o jogo. Várias pessoas encontram-se associadas ao projecto tendo criado vários *bots* diferentes para o jogo, desde *bots* que procedem de forma puramente aleatória, *bots* que apenas defendem e *bots* capazes de vencerem jogadores humanos. Aqui se apresentam alguns dos *bots* já criados:

- *Holdbot* [7];
- *Randbot* [7];
- *DumbBot* [7];
- *Selfbot;*
- *Consbot;*
- *BlabBot* [8];
- *Project20M;*
- *Diplobot;*
- *HaAI;*
- *KissMyBot* [9];
- *Albert* [9];
- *Ruby Electronic Diplomat (R.E.D.);*
- *Minerva;*
- *TheDiplominator;*
- *Brutus* [10].

Alguns dos *bots* agem de forma simples com por exemplo o *Holdbot* que apenas se defende, ou o *Randbot* que actua de forma aleatória.
Outros *bots* alcançam um nível de eficiência mais elevado, como por exemplo o *Project20M*, produzido para um trabalho académico, que era o mais eficiente *bot* existente no início de 2005.

1.4 Estrutura da Dissertação

Este documento encontra-se dividido em 7 capítulos. O primeiro capítulo contém uma introdução descrevendo os objectivos e pressupostos do trabalho, e descrevendo os conceitos do jogo *Diplomacy*. No segundo capítulo procede-se à descrição da estrutura do software desenvolvido e seus componentes. Os 4 capítulos seguintes tratam, detalhadamente, cada um dos módulos do software. O capítulo 3 descreve o sistema de planeamento e respectiva análise algorítmica. O quarto capítulo apresenta a estrutura da base de dados. O capítulo 5 descreve os princípios subjacentes ao sistema de enumeração e o capítulo 6 apresenta as ferramentas usadas na construção de um sistema de teste. Por fim, os capítulos 7 e 8 fornecem conclusões e sugestões para o futuro.

2

Estrutura do Software

Neste capítulo pretende-se fornecer uma descrição do software desenvolvido e a sua estrutura de forma a mais claramente se perceber a abordagem tomada na implementação do mesmo. Todos os diagramas referenciados neste capítulo encontram-se em anexo neste documento. Sendo desenvolvido com recurso à linguagem de programação Java, foi necessário pensar o software como sendo uma estrutura de classes que permitisse a sua implementação segundo os princípios de programação orientada a objectos. A base de dados de movimentos foi implementada através de uma biblioteca de ficheiros. Adicionalmente foi desenvolvida uma aplicação de teste. O diagrama 1 apresenta o diagrama de classes do software.

O software encontra-se dividido nos seguintes quatro pacotes principais:

- *Algorithm* – Este pacote contém todas as classes pertencentes à base de dados, representação de mapas e de movimentos, algoritmo de planeamento e tipos de dados criados. Este pacote encontra-se ainda subdividido nos seguintes pacotes:
    - *Actions* – Neste pacote estão as classes para representação de movimentos;
    - *DataBase* – Neste pacote encontram-se as classes relativas à base de dados;
    - *DataTypes* – Este pacote contém os enumerados dos tipos de dados criados;
    - *Main* – Neste pacote estão as classes relativas ao algoritmo de planeamento;
    - *World* – Este pacote contém as classes para a representação de mapas.
- *Exterior* – Neste pacote estão contidas as classes necessárias ao sistema de enumeração de situações de jogo. Dentro deste pacote existe ainda o pacote *Generators* que contém algumas classes com mecanismos de contagem decimal e binária utilizados no processo de enumeração;
- *Interface* – Este pacote contém a *API* do software;
- *Tester* – Por fim, este pacote contem as classes referentes à *GUI* para testes e interacção com o software *GraphViz*.

A descrição completa e detalhada de cada classe e dos seus conteúdos pode ser facilmente acedida a partir da documentação *JavaDoc* incluída no CD deste trabalho.

2.1 Estrutura da API

A API do software é constituída por uma interface, implementada por uma classe, que fornece mecanismos de arranque, utilização e encerramento do software. Esta interface é responsável pela criação da entidade de gestão da base de dados (objecto da classe *DBManager*) e da entidade de planeamento (objecto da interface *Leadership*). Esta entidade de planeamento não possuí conhecimento directo do algoritmo de planeamento, delegando essa responsabilidade para um componente da interface *Algorithm*.

A estruturação destes componentes em interfaces permite oferecer uma maior noção de modularidade no software, permitindo que facilmente se possa criar uma implementação alternativa de uma destas interfaces para substituir a implementação existente. O diagrama 2 apresenta o diagrama de classes destes componentes.

2.2 Estrutura de um Mapa

O processo de representação de um mapa sob a forma de uma estrutura de armazenamento de dados obriga a que se faça uma reflexão sobre as técnicas já estudadas para implementação de grafos, enquanto estrutura de armazenamento de dados. Para tal é necessário considerar um grafo enquanto estrutura composta por dois elementos essenciais, nós e arestas. No contexto de um mapa Diplomacy, os nós representam regiões e as arestas representam fronteiras entre as regiões. Dito isto, existem dois aspectos fundamentais a considerar para uma correcta implementação deste tipo de estrutura de armazenamento de dados:

- O conjunto de nós do grafo;
- O conjunto de arestas do grafo.

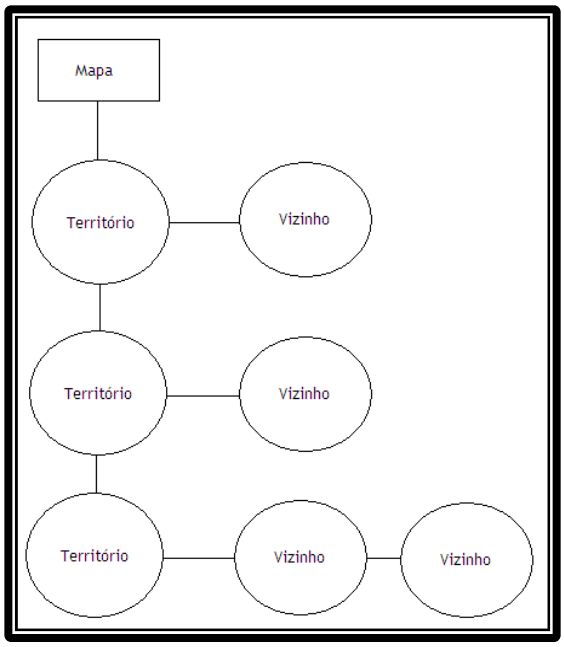


Figura 2.1 - Estrutura de um mapa

Em qualquer uma das técnicas já estudadas, recorre-se a uma lista para armazenar o conjunto de nós e os seus atributos. Em termos de arestas existem duas técnicas principais, a utilização de uma matriz de adjacências ou a utilização de uma lista de adjacências como atributo de cada nó. A matriz de adjacências é útil para grafos densamente conexos. Por outro lado, as listas de adjacência são mais vantajosas para grafos mais dispersos. Neste trabalho, foi decidido utilizar listas de adjacências nos vários nós. Assim sendo, a estrutura de dados de um mapa é basicamente composta por um objecto da classe *Map* que contem métodos para efectuar alguns cálculos e pesquisas no mapa bem como um método para fornecer acesso à lista de nós. Essa lista é construída com base na classe *LinkedList*, fornecida pela linguagem Java, e na classe *Territory*. A combinação destas duas classes permite criar uma lista de objectos da classe *Territory*, sendo que cada objecto representa uma região do mapa e contém os seus atributos, incluindo a referência para um objecto da classe *Unit*, caso exista uma unidade posicionada nessa região, e a referência para a lista de adjacências construída com base na combinação das classes *LinkedList* e *Neighbour* sendo que cada objecto da classe *Neighbour* representa uma fronteira entre a região que referencia a lista e uma outra região do mapa. Note-se que para cada adjacência *B* existente na lista da região *A*, existe uma adjacência *A* na lista da região *B*. A adjacência *B* é constituída pelo identificador numérico da região *B* e por um valor enumerado pela classe Directions que indica se *B* está a uma distância menor, maior ou igual, da região a conquistar, que *A*. A figura 2.1 apresenta, de forma gráfica, um possível estado de um mapa com três regiões. O diagrama 4 apresenta o diagrama de classes do pacote *World* onde se podem observar os detalhes das classes já mencionadas.

2.3 Estrutura de uma Solução

A implementação de uma estrutura de dados para armazenar e executar operações sobre uma solução de um mapa apresenta uma diferença em relação à estrutura analisada anteriormente, apesar de também se recorrer à classe *LinkedList* para a construção da estrutura pretendida. A diferença reside no facto de uma solução ser composta por um conjunto de instruções/movimentos, o que significa que, conceptualmente, uma solução é apenas uma simples lista de instruções e não uma estrutura complexa em forma de grafo. Dito isto, uma solução é composta por um objecto da classe *Movements* que contem a referência para a lista de objectos da classe *Order*, sendo que cada objecto contém os atributos de uma instrução, e disponibiliza vários métodos para efectuar operações e pesquisas sobre a lista. Na figura 2.2 é apresentado um esquema simples de uma possível solução composta por três instruções. O diagrama 5 apresenta o diagrama de classes do pacote *Actions* e expõe os detalhes destas classes.

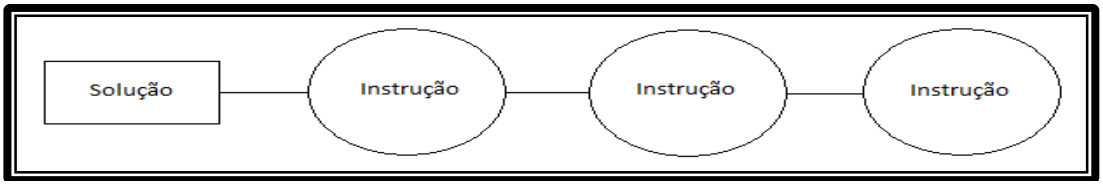


Figura 2.2 - Estrutura de uma solução

3

Sistema de Planeamento

Neste capítulo iremos estudar e descrever o funcionamento e princípios por detrás do sistema de planeamento de movimentos, sendo esta a parte mais importante deste trabalho. Para efeitos de clareza na compreensão do documento, a região que se pretende conquistar no mapa, será denominada por “região alvo”.

3.1 Sub-mapa de Planeamento

Para se obter um sistema de planeamento o mais eficiente possível, é necessário encontrar princípios que permitam uma redução do universo de situações possíveis. Um desses princípios passa por determinar quais as unidades e regiões que podem ter influência no objectivo, que depende do número de turnos ou horizonte temporal do plano. Qualquer unidade ou região que esteja demasiado distante da região alvo não terá qualquer influência no planeamento, devendo ser desprezada. Isto significa que, para se efectuar um planeamento para um número suficientemente reduzido de turnos, poderá ser desnecessário considerar todo o mapa do jogo. Nestas situações, o processamento de um sub-mapa, contendo apenas os elementos relevantes, poderá aumentar a eficiência do planeamento. A questão que se coloca é, como determinar quem tem, ou não, influência no planeamento?
Para responder a esta questão, consideram-se os seguintes requisitos:

- a região alvo é sempre considerada relevante para o planeamento, quer esteja ocupada por uma unidade, ou não;
- a distância de uma unidade ou região é calculada com base na classificação das regiões por níveis de profundidade, sendo que a região alvo pertence sempre ao nível zero, as regiões adjacentes a esta pertencem ao nível um, e assim por diante;
- uma região desocupada só é relevante se a mesma fizer parte de um possível percurso entre uma unidade e a região alvo;
- uma região desocupada que se encontre na extremidade do sub-mapa não faz parte de nenhum percurso entre uma unidade e a região alvo, sendo irrelevante;

- uma unidade só é relevante se a mesma se encontrar a uma distância que lhe permita, recorrendo ao conjunto de movimentos descritos no capítulo 1, aproximar-se o suficiente da região alvo, no espaço de tempo dado pelo número de turnos do planeamento, para poder participar no processo de controlo dessa região, quer seja atacando/defendendo, oferecendo suporte ou cortando suporte;
- cada unidade só pode realizar um movimento em cada turno;
- os movimentos "MOVE" e "CONVOY" são os únicos que permitem que uma unidade se desloque no mapa. Por isso, qualquer unidade que necessite de se deslocar no mapa deverá executar um destes movimentos assim que possível;
- o movimento "MOVE" só pode ser executado para uma região adjacente (a um nível) à região ocupada pela unidade, ou seja, com este movimento, a unidade pode reduzir a sua distância, para a região alvo, em apenas um valor de cada vez;
- o movimento "CONVOY" permite que uma unidade terrestre seja transportada, por uma unidade marítima, para uma região a dois níveis da sua, reduzindo a sua distância em dois valores de cada vez;
- qualquer um dos movimentos possíveis só pode ser executado se não colocar a unidade numa posição na qual perca a sua influência no planeamento.

Após a reflexão sobre os requisitos acima apresentados, é necessário analisar e sistematizar a "situação de batalha" caracterizada pelos movimentos possíveis das unidades existentes nas regiões mais próximas da região alvo:

- para qualquer unidade na região alvo, região de nível 0, o único movimento relevante será o movimento de defesa "HOLD", pois o objectivo desta unidade é defender a região que ocupa;
- as unidades nas regiões de nível 1 tentarão efectuar ataques ("MOVE") à região alvo, bem como oferecer suporte atacante ou defensivo ("SUPPORTMOVE" ou "SUPPORTHOLD");
- as unidades nas regiões de nível 2 tentarão atacar ("MOVE") as unidades nas regiões de nível 1 para cortar o seu suporte.

Com base nisto, determina-se a importância das unidades e regiões da seguinte forma:

- para o último turno do processo de planeamento, qualquer unidade que não esteja numa região pertencente aos primeiros três níveis do mapa (nível 0, nível 1 e nível 2) não poderá participar na batalha;

- para os outros turnos, apenas são importantes, unidades que tenham possibilidade de se posicionarem nestes três níveis, ou que tenham possibilidade de impedir unidades inimigas de o fazerem.

Falta apenas formalizar estas conclusões do ponto de vista matemático. Para tal considera-se o caso limite de planeamento em um turno e o caso geral de planeamento em N turnos.

Para o planeamento em um turno, aplica-se apenas a situação do último turno, ou seja, apenas os primeiros três níveis (ou $2 * 1 + 1$ níveis) são considerados. Para o planeamento em N turnos, aplica-se a situação de aproximação das unidades durante $N - 1$ turnos, seguindo-se a situação do último turno. Durante a aproximação, cada unidade só conseguirá aproximar-se, no máximo, dois níveis em cada turno (caso extremo com utilização sucessiva do movimento "CONVOY"). E só poderá ser impedida por unidades que já estejam mais próximas da região alvo do que ela. Isto significa que, por cada um destes $N - 1$ turnos, será necessário considerar mais dois níveis adicionais no sub-mapa $[2 * (N - 1)]$. Com base nisto, o número máximo de níveis a serem considerados no sub-mapa é dado pela expressão:

$$2 * 1 + 1 + 2 * (N - 1) = 2 * (1 + N - 1) + 1 = 2N + 1$$

Dito isto, qualquer unidade ou região que não se encontre dentro destes $2N + 1$ níveis, não deverá fazer parte do sub-mapa fornecido ao processo de planeamento.

3.2 O Princípio Min-Max

O *Min-Max* [13] é um princípio ou regra de decisão utilizado nas áreas de *Teoria de Decisão*, *Teoria de Jogos* [17] e *Estatística* que se baseia na minimização da perda possível numa decisão enquanto, simultaneamente, se procura maximizar o ganho possível. Este princípio foi originalmente formulado para tratamento do problema de *Soma-Zero* de dois *Jogadores* na área de *Teoria de Jogos* pelo que se torna necessário analisar um pouco este problema e a respectiva área científica.

A *Teoria de Jogos* é um ramo da área de *Matemática Aplicada* que visa a captura e tratamento matemático do comportamento humano em situações estratégicas, denominadas por *Jogos*, nas quais o ser humano necessita de tomar uma decisão. O problema de *Soma-Zero* de dois *Jogadores* não é mais do que um jogo específico no qual participam dois *Jogadores*, ou seja, duas entidades adversárias com capacidade para efectuarem decisões, e no qual o sucesso de um dos *Jogadores* é obtido à custa da perda do outro.

É neste campo que surge o princípio *Min-Max* dizendo formalmente que para qualquer *Jogo de Soma-Zero* de dois *Jogadores* com um número finito de estratégias existe um valor V e uma estratégia mista para cada *Jogador* tal que:

a) Dada a estratégia do *Jogador 2*, o melhor ganho possível para o *Jogador 1* é V;
b) Dada a estratégia do *Jogador 1*, o melhor ganho possível para o *Jogador 2* é –V.

Por outras palavras, a estratégia do *Jogador 1* garante-lhe um ganho no valor de V e um ganho de –V para o *Jogador 2* independentemente da estratégia deste último.

Assim sendo, o algoritmo *Min-Max* resume-se a analisar, de forma recursiva (ver figura 3.1), as combinações possíveis de estratégias de ambos os *Jogadores*, só possível por existirem um número finito de estratégias para ambos, de forma a obter o valor V que minimiza e perda do nosso *Jogador* e, por ser um *Jogo* de apenas dois *Jogadores*, maximiza o seu ganho.

Existem variantes do princípio *Min-Max* para vários *Jogos* possíveis, incluindo *Jogos* de *N Jogadores* nos quais só é possível garantir a minimização da perda e não a maximização do ganho.

Dito isto, para efeitos de compreensão da análise que se irá efectuar em seguida, define-se neste trabalho que:

- Jogo – significa uma situação estratégica de tomada de decisão;
- partida – refere-se ao jogo *Diplomacy*;
- Jogador – refere-se a uma das entidades envolvidas num *Jogo*;
- participante – refere-se aos participantes num jogo *Diplomacy*;
- estratégia individual – conjunto de movimentos, um por cada unidade, para todas as unidades de um dos participantes;
- estratégia global – conjunto de estratégias individuais, uma para cada participante.

3.3 Aplicação do Algoritmo

Para se iniciar a aplicação do algoritmo *Min-Max*, torna-se necessário verificar as condições necessárias à sua aplicação, ou seja, torna-se necessário garantir que as fases de movimentação, descritas no capítulo 2, realizadas numa partida *Diplomacy* constituem *Jogos* de *Soma-Zero* de dois *Jogadores* com um número finito de estratégias possíveis.

Uma fase de movimentação representa uma situação estratégica e portanto um *Jogo*. Qualquer ganho territorial de um participante se torna numa perda imediata para todos os outros, sendo então um *Jogo* de *Soma-Zero*. Como cada unidade pode executar apenas uma de cinco instruções diferentes comprova-se a existência de um número finito não só de estratégias individuais como, consequentemente, também de estratégias globais possíveis, tal que:

Sendo j o número de participantes;
Sendo u_i o número de unidades do participante i;
Sendo U o número total de unidades no mapa tal que $U = \sum_{i=1}^{j} u_i$;
Tendo cada unidade, no máximo, M movimentos possíveis, com $M \in \mathbb{N}$;
Sendo e_i o número máximo de estratégias individuais possíveis do participante i;
Sendo E o número máximo de estratégias globais possíveis tal que $E = \prod_{i=1}^{j} e_i$; (3.1)

Conclui-se que o número máximo de estratégias individuais por participante é dada por:

$$e_i = M^{u_i} \quad (3.2)$$

pois cada movimento da primeira unidade tem que ser combinado com os M movimentos possíveis da segunda unidade e assim por diante, o que dá origem a Arranjos com Repetição. Pelo mesmo princípio de Arranjos com Repetição, se forem consideradas todas as unidades presentes no mapa, independentemente do participante a que pertencem, obtemos a expressão:

$$E = M^{U} \quad (3.3)$$

como sendo o número máximo de estratégias globais do mapa. Assim, para que as equações 3.1 e 3.3 estejam ambas correctas, temos que demonstrar que $M^U = \prod_{i=1}^{j} e_i$. Para isso toma-se a equação 3.1 como ponto de partida para se obter a equação 3.3 da seguinte forma:

$$E = \prod_{i=1}^{j} e_i = \prod_{i=1}^{j} M^{u_i} = M^{u_1} * M^{u_2} * \dots * M^{u_j} = M^{u_1+u_2+\cdots+u_j} = M^{\sum_{i=1}^{j} u_i} = M^U$$

Existe ainda a questão de cada unidade poder executar o mesmo movimento para cada uma das regiões adjacentes o que aumenta o número de estratégias possíveis (quer individuais

quer globais). No entanto, é definido neste trabalho que existe um valor *m* que representa o número máximo de fronteiras que cada região pode ter, portanto:
Sendo m o número máximo de fronteiras de cada região;
Considerando $\overline{m}$ o número médio de fronteiras de cada região;
O número médio de estratégias globais será $\overline{m}M^U$ enquanto que o número máximo de estratégias globais, que representa o pior caso de esforço computacional no qual todas as regiões têm m fronteiras, será mM^U.
Assim sendo, a única condição que falta verificar para se poder aplicar o algoritmo *Min-Max* é a questão da presença de apenas dois *Jogadores* no *Jogo* de *Soma-Zero*. À primeira vista parece que o facto de o *Diplomacy* poder ser jogado por um máximo de sete participantes coloca entraves à aplicação deste algoritmo. No entanto, pode-se facilmente obter os dois *Jogadores* pretendidos se agruparmos os participantes em dois grupos distintos, o grupo de aliados e o grupo de inimigos. Desta forma obtém-se os dois *Jogadores* pretendidos para validar todas as condições necessárias para um *Jogo* de *Soma-Zero* de dois *Jogadores*. Poderá dizer-se que no decorrer de um jogo poderão existir nações neutrais entre si o que originaria um terceiro grupo que invalidaria a aplicação do algoritmo. Tal é verdade mas, para efeitos de realização deste trabalho, foi decidido que o grupo Neutro seria ignorado neste trabalho, sendo uma questão a tratar no futuro.
Após esta validação, surge a necessidade de perceber, no contexto de uma partida *Diplomacy*, o que significa o valor V. Como o *Diplomacy* se trata da conquista de regiões, nomeadamente os *Supply Centres* referidos no capítulo 1, o valor V refere-se naturalmente ao controlo ou posse de uma região alvo, enquanto que –V refere-se ao não controlo dessa mesma região. Deste modo o algoritmo *Min-Max* é aplicado, segundo a sua definição, de forma a determinar qual a estratégia do *Jogador Aliado* que lhe permite ocupar a região V independentemente daquilo que o *Jogador Inimigo* faça.

3.4 Implementação do Algoritmo

De forma a proceder à implementação do algoritmo foi necessário identificar qual a melhor forma para aplicar o planeamento recursivo. Visto que o objectivo era o planeamento com vista à obtenção de uma região específica no final de N fases de movimentação, denominadas neste trabalho por turnos, sendo que em cada turno todas as unidades

executam as suas instruções de forma simultânea, foi definida uma implementação na qual cada passo recursivo analisa um conjunto de movimentos fornecidos em quatro passos:

- **Verificação** – o conjunto de movimentos é analisado para ver se faz sentido de acordo com as regras do *Diplomacy* (exemplo: uma unidade executa um SUPPORTHOLD para uma região na qual não existe nenhuma unidade a executar um HOLD provocando uma situação inválida);
- **Execução** – o conjunto de movimentos é aplicado ao mapa fornecido;
- **Preparação** – o mapa modificado é fornecido a um gerador para o mesmo fornecer a lista de conjuntos de movimentos possíveis
- **Invocação** – são invocadas as chamadas recursivas para processarem o turno seguinte, uma por cada conjunto de movimentos, analisando qual a chamada que consegue fornecer o controlo da região alvo (valor V).

Assim, obtém-se uma árvore de pesquisa recursiva na qual cada nó da mesma representa um conjunto de movimentos das unidades, sendo que o nível de profundidade do nó na árvore representa o turno processado, ver figura 3.1. Quando cada nó do último turno é processado, verifica-se se foi obtida a região alvo.

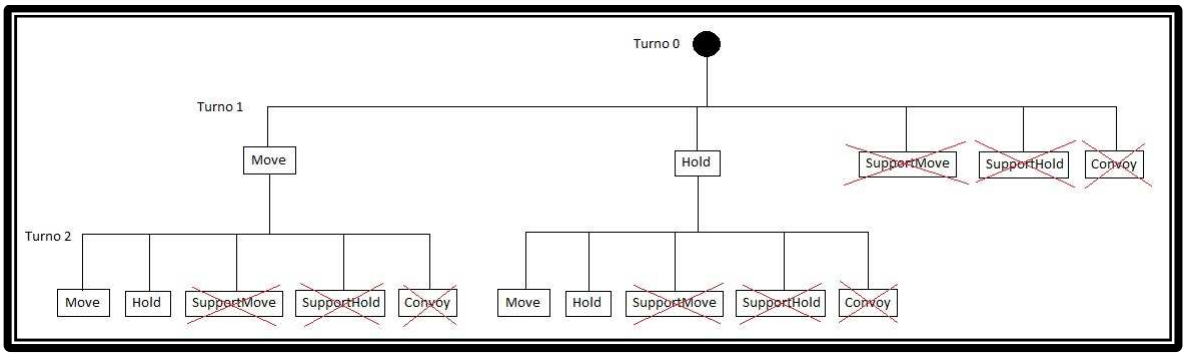


Figura 3.1 - Esquema do planeamento de forma recursiva para um mapa com apenas uma unidade

O processo de *Verificação* torna-se extremamente importante para limitar o volume de expansão da árvore, contribuindo para a redução dos custos computacionais do algoritmo. Caso um determinado nó da árvore de recursividade contenha um conjunto de movimentos inválido, isso significa que não só o processamento desse nó é interrompido, como também toda a sub-árvore de recursividade desse nó para adiante não é gerada, pois se o conjunto de movimentos do turno i é inválido, também é inválido qualquer conjunto de movimentos do turno $i + x, x > 0$, aplicado após o conjunto inválido. Assim sendo, quanto mais exigente e profundo for o processo de *Verificação* mais conjuntos inválidos serão

detectados e mais se reduzirá a quantidades de nós da árvore de recursividade que necessitarão serem processados. Torna-se portanto pertinente analisar as regras de validação às quais cada conjunto de movimentos é submetido.

Assim sendo, um conjunto de movimentos é inválido se:

- Caso se encontre alguma unidade na região alvo e essa unidade não execute uma instrução *HOLD* visto que essa unidade tem que se limitar a defender a sua posição;
- Caso uma unidade faça uma instrução *SUPPORTMOVE* para uma região sem existir nenhuma unidade a realizar uma instrução *MOVE* para a mesma;
- Caso uma unidade faça uma instrução *SUPPORTMOVE* para uma região para a qual a única unidade a realizar uma instrução *MOVE* pertença ao grupo adversário;
- Caso uma unidade faça uma instrução *SUPPORTHOLD* para uma região sem existir nenhuma unidade a realizar uma instrução *HOLD* para a mesma;
- Caso uma unidade faça uma instrução *SUPPORTHOLD* para uma região para a qual a única unidade a realizar uma instrução *HOLD* pertença ao grupo adversário;
- Caso uma unidade realize uma instrução *HOLD* estando numa região a uma profundidade superior a duas vezes o número de turnos, significando que qualquer instrução de *HOLD* retiraria à unidade qualquer influência nos turnos seguintes para efeitos do controlo da região alvo, por ficar a uma distância demasiado elevada dessa região;
- Caso uma unidade execute uma instrução MOVE para uma região estando outra unidade, pertencente ao mesmo grupo, a executar a mesma instrução;
- Caso uma unidade realize uma instrução *CONVOY* entre duas regiões não costeiras;
- Caso uma unidade realize uma instrução *CONVOY* sem existir nenhuma unidade a desejar ser transportada;
- Caso a diferença entre a soma dos pesos dos dois grupos contabilizados numa qualquer região para a qual exista pelo menos uma instrução MOVE ou uma instrução HOLD seja maior que 1, pois tal significa que existem uma ou mais unidades a executarem instruções de suporte desnecessárias.

3.5 Complexidade do Algoritmo

De forma a se proceder à análise da complexidade do algoritmo, é necessário ter em consideração que o número de fronteiras de cada região pode ser variável, assim como a quantidade de unidades em jogo cada momento. No mapa padrão, o número máximo de unidades é de 34 devido ao facto de só poder existir uma unidade por cada *Supply Centre* (capítulo 1). Devido à dinâmica destes factores será desejável realizar uma análise que permita obter resultados para o pior cenário possível assim como para o cenário mais provável. É necessário também ter em conta que qualquer medida de desempenho depende sempre da capacidade do hardware sobre o qual é executado o software.
Assim sendo, considere-se o seguinte:

Seja t o tempo médio gasto no processamento de cada instrução, em segundos;
Seja m o número máximo de fronteiras de cada região e $\overline{m}$ o seu valor médio;
Seja M o número máximo de movimentos possíveis. No *Diplomacy* $M = 1 + 4m$;
Seja U o número máximo de unidades no mapa, sendo que no *Diplomacy* $U = 34$;
Seja $\overline{u}$ o número médio de unidades no mapa;
Seja n o número de turnos processados;

Como verificado neste capítulo, segundo a equação 3.2, o número de conjuntos de movimentos possíveis é igual à quantidade de instruções elevado ao número de unidades existentes no mapa.
Em primeiro lugar é importante analisar o cenário mais provável, ou seja, o cenário composto pelos valores médios dos parâmetros variáveis. Assim:
Seja α o número de conjuntos de movimentos possíveis considerando o valor médio do número de fronteiras de cada região e o número médio de unidades no mapa, obtém-se $\alpha = \overline{m}M^{\overline{u}} = \overline{m}(1 + 4m)^{\overline{u}}$ instruções. O seu tempo de execução será $\alpha t = \overline{m}(1 + 4m)^{\overline{u}}t$.
Considere-se $\overline{c}$ a complexidade do cenário mais provável. Ora a complexidade é obtida a partir da soma dos processamentos de α ao longo do número de turnos desejado da seguinte forma:

- no primeiro turno α é processado uma vez;
- no segundo turno α é processado uma vez por cada elemento de α no turno anterior, ou seja, α é processado α vezes, obtendo-se $\alpha * \alpha = \alpha^2$;
- no turno n α é processado α^n vezes.

Sendo $\overline{c}$ a soma de todos estes passos, obtém-se

$\overline{c} = \sum_{i=1}^{n} \alpha^i = \sum_{i=1}^{n} \left(\overline{m}(1+4m)^{\overline{u}}\right)^i = \sum_{i=1}^{n} \overline{m}^i (1+4m)^{\overline{u}i}$ instruções. (3.4)

Considerando o tempo t, obtém-se

$\overline{c} = t\sum_{i=1}^{n} \alpha^i = t\sum_{i=1}^{n} \left(\overline{m}(1+4m)^{\overline{u}}\right)^i = t\sum_{i=1}^{n} \overline{m}^i (1+4m)^{\overline{u}i}$ segundos. (3.5)

Pode-se então concluir que a complexidade do algoritmo aumenta de forma directamente proporcional ao número médio de fronteiras de cada região, e aumenta de forma exponencial com o aumento de unidades e/ou turnos considerados.

Para analisar o pior cenário possível é necessário considerar a existência de todas as 34 unidades possíveis no mapa, bem como é também necessário assumir que todas as regiões possuem m fronteiras o que significa que o mapa contém o número máximo de fronteiras possíveis. Nesse caso:

Seja β o número de conjuntos de movimentos possíveis considerando o pior cenário descrito acima, obtém-se:

$\beta = mM^U = m(1+4m)^{34}$ instruções ou

$\beta t = mM^U = m(1+4m)^{34}t$ segundos.

Sendo C a complexidade do pior caso e pelo o mesmo raciocínio usado nas equações 3.4 e 3.5, obtém-se:

$C = \sum_{i=1}^{n} \beta^i = \sum_{i=1}^{n} m(1+4m)^{34^i} = \sum_{i=1}^{n} m(1+4m)^{34i}$ instruções. (3.6)

$C = t\sum_{i=1}^{n} \beta^i = t\sum_{i=1}^{n} m(1+4m)^{34^i} = t\sum_{i=1}^{n} m(1+4m)^{34i}$ segundos. (3.7)

Após a obtenção destas expressões resta apenas considerar o facto de o jogo *Diplomacy* tradicional apresentar sempre o mesmo mapa, apesar de algumas versões digitais fornecerem outros mapas para além do mapa tradicional, o que significa que se pode considerar os valores m e $\overline{m}$ como sendo constantes devido ao mapa não sofrer alterações. Se se considerar que o hardware presente no computador é sempre o mesmo, significa que t também pode ser considerado um valor constante pelo que os parâmetros com mais influência no aumento do tempo de execução do algoritmo são o número de unidades e de turnos considerados, ao provocarem um crescimento exponencial da complexidade. Pelo que se torna computacionalmente dispendioso executar o algoritmo para cenários com muitas unidades e e/ou turnos. Na tabela 3.1 são apresentados tempos de execução teóricos, em segundos, para o cenário mais provável $\overline{c}$, com base na equação 3.5, com

alguns valores propostos em termos de número de unidades e turnos, assumindo um tempo médio $t = 0{,}001$ segundos e um número médio e máximo de 3 fronteiras por cada região.

turnos \ *unidades*	1	5	10	20
1	3,900E-02	1,114E+03	4,136E+08	5,701E+19
2	5,460E-01	4,136E+08	5,701E+19	1,084E+42
3	7,137E+00	1,536E+14	7,860E+30	2,059E+64
4	9,282E+01	5,702E+19	1,084E+42	3,914E+86

Tabela 3.1 - Tempos de execução teóricos do algoritmo Min-Max, em segundos

Por fim, é extremamente importante realçar a relevância do processo de *Verificação* de cada conjunto de movimentos descrito neste capítulo por ser o grande mecanismo redutor do tempo de execução do sistema de planeamento. Como em cada ramo que se percorre na árvore de recursividade é necessário processar todos os $\overline{m}(1+4m)^{\overline{u}}$ conjuntos de movimentos e fronteiras possíveis, pode-se pensar no custo computacional do algoritmo como sendo a soma do factor $\overline{m}(1+4m)^{\overline{u}}$ tantas vezes quantos os ramos percorridos na árvore de recursividade. Ora sabendo que a detecção de um conjunto de movimentos inválido permite ignorar toda a possível sub-árvore a partir do nó actual, isso significa que se ignoram um ou mais factores $\overline{m}(1+4m)^{\overline{u}}$, reduzindo o custo computacional do algoritmo. Isto significa que quanto maior for a percentagem de nós da árvore de pesquisa ignorados, maior será a redução do custo computacional do algoritmo. Se o processo de Verificação permitir ignorar 40% a 50% da árvore de pesquisa, isso resultará numa grande redução do custo computacional.

Estes factos justificam o esforço gasto na criação de um processo de *Verificação* exaustivo para conseguir optimizar o esforço computacional do algoritmo *Min-Max*.

A tabela 3.2 apresenta valores experimentais para um número médio de 3 fronteiras por região. Registaram-se tempos inferiores aos tempos da tabela 3.1, o que indica que o sistema computacional usado consegue obter um menor valor t.

turnos \ unidades	1	2
1	9,00E-03	1,50E+01
2	2,00E-02	1,60E+01

Tabela 3.2 - Tempos de execução experimentais do algoritmo Min-Max, em segundos

4

Base de Dados

A inclusão de um mecanismo de armazenamento de informação em forma de base de dados permite desenvolver no agente de inteligência artificial a capacidade de aprendizagem ao longo do tempo permitindo-lhe responder mais rapidamente a situações já resolvidas anteriormente. Para tal é necessário desenvolver um processo que permita comparar a situação actual com as situações já armazenadas de forma eficiente. Para além disso é também necessário criar processos para adicionar, carregar e salvaguardar a informação.

Neste capítulo irá proceder-se à análise e exposição da estrutura da base de dados e das suas ferramentas.

4.1 Estrutura da Informação

No que toca à estruturação da informação da base de dados é necessário analisar quer a sua estrutura em ficheiros quer a sua estrutura enquanto se encontra em execução no sistema. No entanto, antes de se proceder a essa análise, importa realçar que este trabalho tem o seu foco principal na questão do planeamento e decisão de jogadas e não se entendeu como necessário proceder à inclusão de mecanismos complexos de bases de dados sendo esta resumida a uma estrutura em forma de tabela com capacidade para armazenar mapas e as respectivas soluções. Igualmente, os seus dados resumem-se a ficheiros com a informação em formato textual sem qualquer mecanismo de protecção ou ocultação da informação.

4.1.1 Estrutura em Ficheiros:

Os ficheiros da base de dados encontram-se localizados numa pasta interna ao software. O conteúdo destes ficheiros é estruturado por uma notação criada pelo autor deste trabalho. A base de dados é constituída por um ficheiro de configuração com o nome *DBconf* no qual se armazena a informação sobre o número de mapas contidos na base de dados, sob a forma de um número inteiro. Cada par [mapa, solução] é armazenado num ficheiro com o

nome *DBline_x*, sendo *x* um identificador numérico correspondente ao número da sua linha na tabela criada em execução, ou seja, o ficheiro *DBline_1* contém o par [mapa, solução] correspondente à primeira linha da tabela em execução e assim por diante. Naturalmente, no primeiro momento de execução do software não existe nenhuma linha na tabela pelo que também não se encontra armazenado qualquer par [mapa, solução] e o ficheiro de configuração contém o número zero. O identificador numérico é um número inteiro sequencial, atribuído de forma automática. Ao se adicionar um par [mapa, solução] à base de dados, o valor do identificador do par anterior é incrementado em uma unidade e atribuído a este par. Ao primeiro par adicionado é atribuído o identificador 1.
De seguida é apresentada a estrutura interna do ficheiro de configuração e de cada ficheiro de linha:

Ficheiro DBconf:

<Número de linhas da tabela>

O ficheiro de configuração contém simplesmente um número indicador da quantidade de ficheiros de linha existentes.

Ficheiro DBline_<x>:

LINE_INFO

<Número de Turnos>

MAP_INFO

<Número de Regiões>

MAP

T <Id. da Região 1> <Tipo da Região 1> <Nível de Profundidade da Região 1>

B <Id. da Fronteira 1> <Direcção da Fronteira 1> ... <Id. da Fronteira N> <Direcção da Fronteira N>

E <Região 1 de Exclusão> ... <Região N de Exclusão>

[UNIT|NO_UNIT]

(<Id. da Unidade> <Tipo da Unidade> <Id. da Região> <Id. da Carga> <Relação>)?

...

T <Id. da Região N> <Tipo da Região N> <Nível de Profundidade da Região N>

B <Id. da Fronteira 1> <Direcção da Fronteira 1> ... <Id. da Fronteira N> <Direcção da Fronteira N>

E <Região 1 de Exclusão> ... <Região N de Exclusão>

[UNIT|NO_UNIT]

(<Id. da Unidade> <Tipo da Unidade> <Id. da Região> <Id. da Carga> <Relação>)?

SOLUTION_INFO

<Número de Instruções>

(SOLUTION)?

*(<Id. da Região Inicial> <Id. da Região Final> <Id. da Unidade> <Id. da Unidade Suportada> <Acção> <Turno>)**

Cada ficheiro de linha contém, em primeiro lugar, o número de turnos a que se refere a respectiva solução. Este valor é armazenado na tabela de forma a servir como mecanismo de filtro no processo de comparação do qual se falará mais adiante. Após este valor, existe um segundo valor que indica o número de regiões que constituem o mapa. De seguida segue-se a representação da informação de cada região do mapa incluindo o tipo de região, as suas fronteira e os territórios que representam situações de exclusão mútua no que toca à localização de unidades caso existam. Para cada região pode ser representada ainda a informação da unidade que a ocupa caso ela exista. Depois da representação do mapa segue-se a representação da quantidade de instruções que fazem parte da solução seguindo-se uma ou mais linhas, uma linha por cada instrução, contendo a representação dos dados de cada instrução como por exemplo as regiões e unidades envolvidas. O ficheiro de linha contém uma e uma só solução para o respectivo mapa.
Toda a informação está devidamente separada e organizada por marcadores descritos em seguida:

LINE_INFO – Início do ficheiro. Após este marcador regista-se o número de turnos;
MAP_INFO – Após este marcador regista-se o número de regiões;
MAP – Após este marcador encontram-se os dados de cada região e unidades do mapa;
UNIT – Indica a presença de uma unidade cujos dados são registados na linha seguinte;
NO_UNIT – Indica a ausência de uma unidade, dispensando a linha com esses dados;
SOLUTION_INFO – Início da solução seguindo-se o número de instruções da mesma;
SOLUTION – Último marcador após o qual se regista cada instrução linha a linha.

4.1.2 Estrutura em Execução:

Em termos de estrutura de código, a base de dados é composta pelas classes *DataLine* e *DBManager* (ver diagrama 3). Os objectos do tipo *DataLine* são simplesmente linhas da tabela que armazenam o mapa, a solução e o número de turnos da solução para serem acedidos pelo objecto da classe *DBManager* o qual se trata de um gestor da base de dados responsável por efectuar o carregamento e salvaguarda da informação aquando da abertura e encerramento do software, bem como por efectuar as operações de adição e comparação de mapas. Todas estas operações são disponibilizadas através de uma interface composta pelos seguintes métodos:

DBManager() – O construtor da classe que procede à criação do objecto e consequente carregamento imediato da informação contida nos ficheiros para a memória.

exit() – Este método é invocado pela *API* do programa antes do seu encerramento de forma a se proceder à salvaguarda da base de dados da memória para ficheiros. É de notar que o gestor da base de dados começa por analisar quais são as linhas da base de dados que já se encontram escritas em ficheiros pois essas não necessitam de serem escritas novamente de forma a ser evitado algum esforço computacional desnecessário.

add(mapa, solução, turnos) – Método que permite adicionar uma linha à base de dados com o par [mapa, solução] fornecidos.

compare(mapa, turnos) – Método para invocar o processo de comparação de mapas. É fornecido o mapa a comparar bem como o número de turnos a que se refere. O número de turnos permite excluir da comparação os mapas da base de dados que se referem a diferentes números de turnos.

4.2 Comparador de Mapas

Nesta secção detalhamos o processo de comparação de mapas. O algoritmo é complexo e tem impacto no desempenho computacional do software desenvolvido neste trabalho. Para efeitos de explicação deste processo, o componente responsável pela comparação de mapas será denominado de Comparador para mais fácil referência.

De forma a minimizar o tamanho da base de dados foi decidido implementar um comparador capaz não só de detectar mapas exactamente iguais entre si mas também mapas equivalentes do ponto de vista da sua solução. Por exemplo, a solução para um mapa com duas unidades aliadas e uma unidade inimiga deverá ser a mesma para um mapa com as mesmas unidades descritas anteriormente, e nas mesmas localizações, mais uma unidade aliada pois, em qualquer dos casos, a diferença de peso na batalha será sempre favorável ao grupo aliado sendo que a unidade aliada em excesso não necessita de ser utilizada na batalha. Se juntarmos esta capacidade ao facto de o número de turnos ser usado para filtrar os mapas armazenados a comparar consegue-se minimizar não só a dimensão total da base de dados como também a dimensão do subconjunto de mapas da mesma a serem comparados com o mapa pretendido.

Para tal, o Comparador começa por verificar se o mapa a comparar se refere ao mesmo número de turnos que o mapa armazenado. Em caso afirmativo procede-se a uma contabilização da quantidade de unidades aliadas e inimigas no mapa para determinar o peso total de ambos os grupos. O mapa que está a ser comparado deverá ter a mesma quantidade de unidades do grupo com menor peso total no mapa armazenado ou não poderá ser considerado equivalente a este. Após esta verificação, o Comparador utiliza um processo recursivo que analisa cada região a partir do ponto central, ou seja, a região de profundidade zero do mapa em direcção às extremidades, ou seja, as regiões de maior profundidade do mesmo. Começando pela região de profundidade zero e procedendo de forma recursiva para cada uma das regiões adjacentes com maior profundidade que a mesma (no caso da região alvo tal é equivalente a todas as regiões adjacentes à mesma), o Comparador verifica se a região faz fronteira com regiões de maior profundidade pois em caso negativo significa que a região analisada se encontra na extremidade do mapa e pode conter uma unidade com possibilidade de ser descartada no caso de existência de excesso de diferença de peso de um dos grupos da batalha. De seguida verifica se a região tem as mesmas características, tipo de região, tipo de unidade se esta existir, etc. da região do mapa armazenado. À medida que o processo recursivo vai encontrando unidades, vai-se incrementando o peso do respectivo grupo (aliado ou inimigo) de forma a encontrar o ponto a partir do qual pode começar a ignorar unidades do grupo com maior peso total.

A título de exemplo, a figura 4.1 ilustra dois mapas que, apesar de conterem um número diferente de unidades, são equivalentes para efeitos de planeamento. As soluções dos dois mapas são iguais pois a unidade adicional, na região 4, que o mapa da direita contém não provoca qualquer alteração ao resultado independentemente dos movimentos que execute.

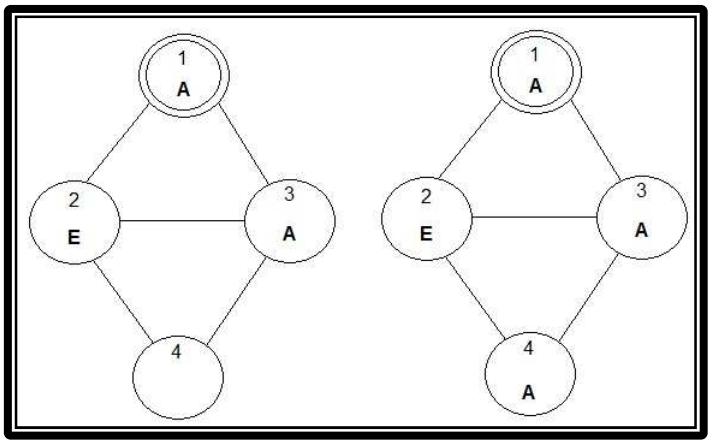


Figura 4.1 - Exemplo de dois mapas equivalentes. A letra A representa a presença de uma unidade aliada na região e a letra E representa uma unidade inimiga.

Analisando formalmente esta questão obtém-se o seguinte:

Seja Pc_{small} o peso, no mapa comparado, do grupo de menor peso total;
Seja Pc_{big} o peso, no mapa comparado, do grupo de maior peso total;
Seja Pa_{small} o peso, no mapa armazenado, do grupo de menor peso total;
Seja Pa_{big} o peso, no mapa armazenado, do grupo de maior peso total.

À media que as unidades vão sendo encontradas pelo processo recursivo, vai-se incrementando Pc_{small} ou Pc_{big} caso a unidade pertença ao grupo de menor ou maior peso total respectivamente.
Quando se atinge o ponto em que $Pc_{small} = Pa_{small} \wedge Pc_{big} = Pa_{big}$, sendo $Pa_{big} = Pa_{small} + 1$, existem duas possibilidades para a próxima unidade encontrada:

- A unidade encontrada pertence ao grupo de menor peso total, fazendo com que $Pc_{small} > Pa_{small}$, o que significa que o mapa comparado é considerado não equivalente ao mapa armazenado e o comparador interrompe o processo recursivo para passar para o próximo mapa da base de dados;
- A unidade encontrada pertence ao grupo de maior peso total, fazendo com que $Pc_{big} > Pa_{small} + 1$, o que significa que o grupo correspondente não necessita desta unidade para ganhar a batalha pelo que os mapas poderão ser equivalentes sendo necessário continuar o processo recursivo.

Após este passo, o comparador tem de estabelecer uma relação de igualdade entre as fronteiras com regiões de maior profundidade de ambos os mapas. Se a região do mapa armazenado contem duas regiões terrestres e uma região marítima de maior profundidade, então será necessário que a região do mapa comparado apresente a mesma situação. Caso contrário os mapas serão considerados diferentes. Se as fronteiras de ambas as regiões forem consideradas iguais o comparador dá continuidade ao processo recursivo invocando-o para cada uma das fronteiras.

Depois de atingir a(s) extremidade(s) do mapa, o processo recursivo retornará os resultados no sentido inverso até se atingir o resultado final. Caso o resultado seja positivo, a solução do mapa armazenado é apresentado como solução do mapa comparado.

O esforço computacional do processo de comparação de mapas está dependente de vários factores difíceis de prever ou quantificar. Entre esses factores contam-se a quantidade de mapas armazenados na base de dados, a dimensão desses mesmos mapas (número de regiões e unidades que possuem) e a posição do mapa pretendido na base de dados. Todo este dinamismo tornou impossível a realização de uma análise formal deste processo dentro dos prazos necessários para a conclusão deste trabalho. No entanto, são de seguida incluídos tempos de execução experimentais para cumprimento de objectivos meramente informativos. Nestas medidas foi considerada uma base de dados com 2 mapas, de dimensões relativamente pequenas, para 1 turno.

posição na BD	tempo	unidades	regiões
primeira	0,00E+00	1	2
última	1,50E-02	3	4

Tabela 4.1 - Tempos de execução do comparador da base de dados

O tempo de execução do comparador para o caso do mapa na primeira posição da base de dados apresentou um valor inferior a 1 milissegundo. A quantidade de unidades e de regiões parecem ser os factores que mais contribuem para o aumento do tempo de execução do comparador.

5

Sistema de Enumeração

Neste capítulo será analisada a estratégia seguida para o desenvolvimento de um algoritmo de enumeração de cenários de jogo. Este sistema permitirá a inicialização da base de dados do software, povoando-a com um conjunto de cenários básicos de jogo, não equivalentes entre si, e respectivas soluções de planeamento.

5.1 Enumeração de Grafos & Isomorfismo

Enumeração de Grafos [19] é um dos estudos das áreas de Combinatória e Teoria de Grafos, ramos da área Matemática, que se baseia no estudo e resolução de problemas de contagem de todos os grafos passíveis de serem construídos com base em alguns parâmetros estabelecidos. Existem vários estudos nesta área sendo George Pólya [21], Arthur Cayley [22] e John Howard Redfield [23] os grandes pioneiros da mesma.

No problema de contagem de grafos surge também a noção de Isomorfismo de Grafos [25]. A definição formal de grafos isomórficos diz que:

Sendo G e H dois grafos, e $V(G)$ e $V(H)$ os conjuntos de vértices desses grafos, existindo uma função bijectiva f tal que: $f: V(G) \rightarrow V(H)$

Então, dois vértices $f(u)$ e $f(v)$ são adjacentes em H se e só se os vértices u e v também são adjacentes em G.

Por outras palavras, é possível obter H a partir de G, em termos gráficos com recurso a rotações de G, pois ambos são equivalentes entre si, ver figura 5.1.

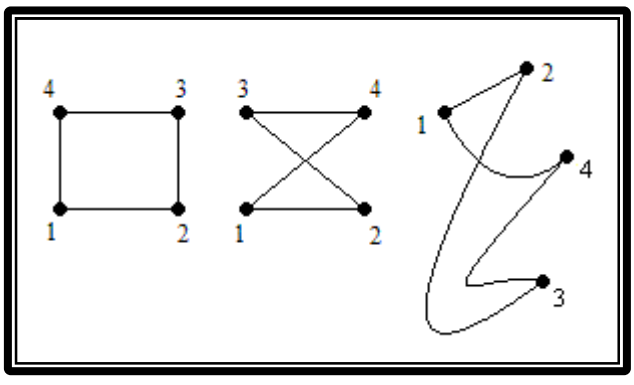
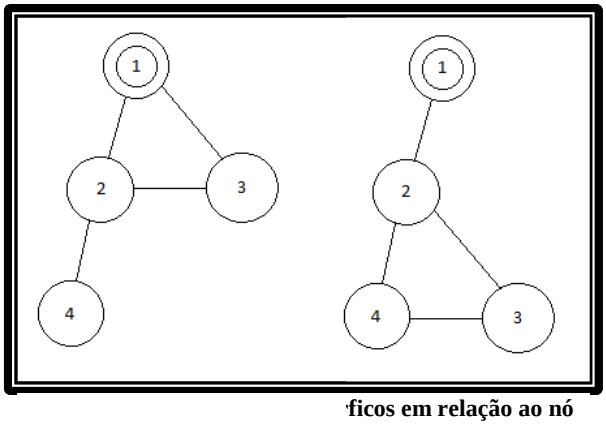


Figura 5.1 - Exemplo de grafos isomórficos

Com base nestas informações, colocam-se três grandes desafios ao desenvolvimento do sistema de enumeração pretendido:

- Em primeiro lugar, apesar de alguns dos estudos já desenvolvidos sobre enumeração de grafos contemplarem a enumeração de grafos não isomórficos, tais estudos assumem o número de nós do grafo como parâmetro de entrada e geram grafos desconexos. Tal representa um problema devido à impossibilidade de um grafo desconexo fazer sentido enquanto representação de um mapa, assim como o facto de existir a necessidade, neste trabalho, de realizar uma enumeração baseada no número de turnos e de fronteiras em vez do número de regiões (nós do grafo);
- Por outro lado, o problema de detecção de isomorfismo entre dois grafos pertence à categoria de Problemas NP, ou seja, problemas cuja solução supõe-se ser impossível de encontrar em tempo polinomial, o que significa que, em termos de esforço computacional, este problema não pode ser resolvido em tempo útil. Este facto não permite assumir uma estratégia de enumeração de todos os grafos possíveis seguida de uma detecção de isomorfismo;
- Há que analisar a questão do isomorfismo no contexto do jogo Diplomacy. Devido à relevância, no grafo, do nó que representa a região alvo, dois grafos considerados isomórficos pela definição formal apresentada anteriormente, poderão não ser isomórficos em relação aos movimentos possíveis para as unidades que se encontram no mapa. A figura 5.2 ilustra dois grafos que são estruturalmente equivalentes mas não são isomórficos. Repare-se que, em termos estruturais, ambos os grafos são constituídos por quatro nós, dos quais três nós formam uma figura triangular e o quarto nó se une a um dos vértices desse triângulo. No entanto, em termos semânticos, os grafos são distintos pois, considerando o planeamento de um turno, no grafo da esquerda uma unidade posicionada na região 3 poderá atacar a região 1 ou oferecer suporte, enquanto que no grafo da direita, a mesma unidade poderá apenas influenciar as regiões 2 e 4.



ficos em relação ao nó lvo

- Por outro lado há que considerar também a questão das regiões marítimas. Ao enumerar várias combinações de regiões terrestres e marítimas possíveis poderão surgir situações de isomorfismo neste aspecto.

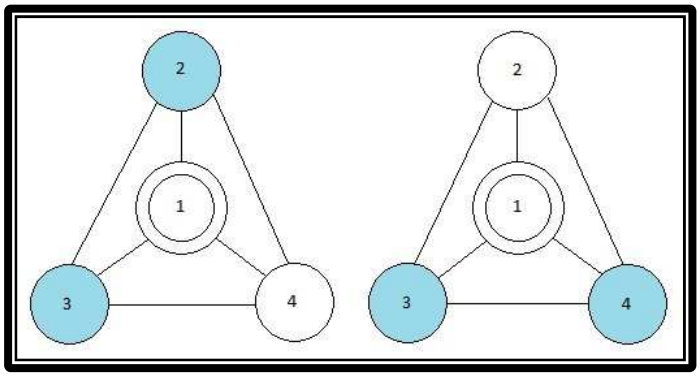


Figura 5.3 - Grafos isomórficos em relação aos tipos de regiões

A figura 5.3 ilustra uma situação na qual as regiões 2 e 4 são diferentes em ambos os grafos. No entanto, os dois grafos são isomórficos pois, em ambos os casos, as duas regiões marítimas são vizinhas das regiões terrestres e vizinhas entre si.

5.2 Estratégia Abordada

Devido a todas as questões referidas neste capítulo, foi decidido abordar o problema com o objectivo de conseguir um processo de enumeração o mais completo possível, garantindo o não isomorfismo entre os vários cenários enumerados, ao invés de tentar alcançar a taxa de 100% de cenários não isomórficos possíveis.

Para tal é necessário proceder da seguinte forma:

- Em primeiro lugar, é necessário gerar combinações de nós em cada um dos níveis de profundidade do grafo com base no número de turnos e número máximo de fronteiras de cada nó;
- Depois é necessário construir ramos/ligações entre os nós. Neste momento é necessário realizar uma distinção entre ligações entre dois nós de profundidades consecutivas, denominadas neste trabalho por ligações verticais, e ligações entre dois nós de igual profundidade, denominadas ligações laterais ou horizontais. Para efeitos de coerência de um mapa, todos os nós têm que apresentar pelo menos uma ligação vertical com um nó de menor profundidade, com excepção do nó referente à região alvo pois esse será o nó de menor profundidade. Assim sendo será necessário gerar combinações apenas de ligações laterais entre os nós;

- De seguida é necessário gerar combinações de nós marítimos e nós terrestres para cada sequência de nós gerada no primeiro passo;
- Por fim é necessário gerar combinações de unidades aliadas e inimigas com base nas sequências de nós, do primeiro passo, e de nós marítimos do terceiro passo.

A cada iteração destes quatro passos é gerado um novo cenário, constituído pelo mapa com as regiões marítimas e terrestres definidas e com uma ou mais unidades posicionadas no mapa, que é inspeccionado, para garantir a limitação do número máximo de fronteiras de cada região, e submetido ao sistema de planeamento para posteriormente ser armazenado na base de dados.

5.3 Geradores

De forma a gerar as combinações de elementos referidas anteriormente, procedeu-se à criação de alguns geradores de combinações numéricas sob a forma de contadores numéricos. O pacote *Generators*, que se encontra dentro do pacote *Exterior*, contém as classes relativas aos três geradores desenvolvidos. Segue-se uma breve explicação sobre cada um deles, podendo recorrer-se à documentação *Javadoc* incluída neste trabalho para mais informações. Os geradores são os seguintes:

- **NodeGenerator** – gera combinações de nós com base no número de turnos e no número máximo de fronteiras de cada nó;
- **ConfigurableGenerator** – gera combinações de ligações laterais ou de nós marítimos e terrestres com base na sequência de nós gerada previamente;
- **UnitGenerator** – gera combinações de unidades com base nas relações e tipos de unidades possíveis e no número de nós da sequência de nós gerada previamente.

Os geradores são simplesmente contadores numéricos cuja base numérica não é uma base fixa (base decimal, base binária, etc.), mas varia consoante a casa numérica considerada, com excepção no *UnitGenerator* cuja base é fixa. Os contadores são representados por estruturas de dados vectoriais, em que o índice do vector indica o nível de profundidade, excepto no *UnitGenerator* no qual o índice representa o identificador numérico de cada região subtraído em uma unidade, sendo os índices pertencentes ao conjunto $C = \{0, 1, \dots, 2t\}$, sendo t o número de turnos definido.

Apesar de se trabalhar com uma base variável, o princípio de contagem é igual ao de um contador de base fixa. Por exemplo, pensando num contador em base decimal até às centenas, ou seja, com duas casas numéricas pode-se estruturar o mesmo de forma vectorial com índices de zero a um em que a posição de índice zero do vector representa as unidades enquanto que a posição de índice um representa as dezenas. Trabalhando em base decimal, ou seja, numa base fixa com dez valores possíveis em cada casa numérica, o processo de contagem passará por incrementar sucessivamente a casa numérica de ordem mais baixa (as unidades) até se atingir o último valor possível para essa casa, situação na qual se incrementa o valor da casa seguinte e se retorna o valor da casa corrente ao valor inicial de forma a recomeçar o processo de incrementação de novo, ver figura 5.4.

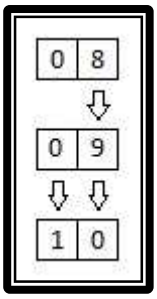


Figura 5.4 - Processo de contagem decimal

Da mesma forma, os contadores desenvolvidos incrementam sucessivamente o valor da casa numérica de ordem mais baixa até que esta atinja o último valor da base numérica correspondente a essa casa numérica específica. A excepção a este procedimento encontra-se no *NodeGenerator* no qual cada casa, após atingir o último valor da sua base retorna para o valor 1 e não para o valor 0 devido ao contexto de jogo em que o contador está inserido. Por exemplo, apresentar o valor 101 num contador *NodeGenerator* de três casas numéricas significaria a criação de um grafo com um nó de profundidade zero e um nó de profundidade dois, o que originaria um grafo desconexo pelo facto de que as ligações verticais serem possíveis apenas entre nós de profundidades consecutivas, ver figura 5.5.

Assim sendo, resta apenas apresentar e explicar as bases usadas em cada contador.

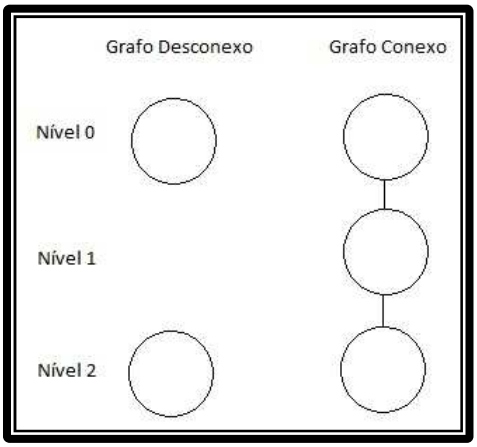


Figura 5.5 - Exemplo de um grafo desconexo

5.3.1 NodeGenerator

No processo de geração de nós é necessário ter em conta a limitação de um máximo de m fronteiras, ou seja, ligações em cada nó. Tal significa que, por cada nó de profundidade i, só poderão existir um máximo de m nós de profundidade $i + 1$, sendo que na profundidade zero só pode existir um nó que representa a região alvo, o que significa que a posição de

índice zero trabalhará em base binária (base 2). Assim sendo obtém-se o seguinte vector de bases numéricas:

$[1+1, m+1, m*m+1, \dots] <=> [m^0+1, m^1+1, m^2+1, \dots, m^{2t}+1]$, sendo t o número de turnos definido.

Pode-se observar que este método dá origem a situações em que um nó tem uma ligação para um nó de profundidade inferior à sua e tem uma ligação para cada um dos m nós de profundidade superior à sua, o que dá origem a um nó com m+1 ligações. No entanto, tal não apresenta um problema devido ao facto de cada cenário ser verificado antes de ser submetido ao processo de planeamento e armazenamento do mesmo. Tais cenários serão descartados aquando da sua verificação. Por forma a encontrar um vector de bases numéricas que não desse origem a esse tipo de cenários seria necessário a adopção de expressões matemáticas cuja complexidade e esforço de implementação não justificaria o ganho em termos de esforço computacional do processo de enumeração.

5.3.2 ConfigurableGenerator

Este gerador é utilizado tanto para gerar massas de água no mapa como também para gerar combinações de ligações laterais. Este gerador apresenta uma forte dependência da quantidade de nós gerados pelo gerador de nós, o que significa que o vector de bases numéricas deste gerador será dado pela sequência de nós gerada a cada iteração do gerador de nós.

5.3.3 UnitGenerator

O gerador de unidades é o único que apresenta uma base fixa definida pela soma da quantidade de tipos diferentes de unidades com a quantidade de possíveis relações entre elas, descartando a relação de neutralidade com já foi referido no capítulo 3, mais o valor um de forma a considerar a situação de ausência de unidade. Assumindo o contexto do jogo Diplomacy sabe-se que:

- Existem dois tipos de unidades (terrestres e marítimas);
- Existem dois tipos de relações (aliados e inimigos);

Portanto, pode-se calcular o valor da base fixa B da seguinte forma:

Sendo u a quantidade de tipos de unidades;

Sendo r a quantidade de relações, descartando a relação neutral;

$$B = u * r + 1 = 2 * 2 + 1 = 5$$

Como a posição de índice i do vector representa o nó com o Id $i + 1$, obtêm-se, para o conjunto de nós do mapa, as Arranjos com Repetição dos seguintes valores:

0 – Nenhuma unidade;
1 – Unidade Terrestre Aliada;
2 – Unidade Marítima Aliada;
3 – Unidade Terrestre Inimiga;
4 – Unidade Marítima Inimiga.

O que perfaz um total de 5^n combinações possíveis, sendo n o número de nós/regiões do mapa.

5.4 Implementação

A implementação do sistema de enumeração foi efectuada com recurso aos princípios e mecanismos descritos neste capítulo. Por forma a superar as condicionantes descritas no secção 5.1, foi seguida a estratégia descrita no secção 5.2 através de mecanismos de geração de elementos de um mapa (secção 5.3) e de um conjunto sucessivo de passos:

1. Em primeiro lugar gera-se uma sequência possível de regiões/nós nos diferentes níveis do mapa, recorrendo a um gerador do tipo *NodeGenerator*. Neste passo, o número de turnos restringe a quantidade de níveis do mapa;
2. De seguida criam-se as fronteiras entre regiões de níveis adjacentes, tendo em conta o número máximo de fronteiras permitidas em cada região. Não são efectuadas permutações das fronteiras entre as regiões de um mesmo nível para se evitar possíveis situações de isomorfismo (secção 5.1);
3. Gera-se uma sequência possível de unidades, incluindo o seu tipo (terrestre ou maritímo), a sua localização no mapa e o seu comportamento (aliado ou inimigo). É utilizado um gerado do tipo *UnitGenerator*;
5 Gera-se uma sequência possível de fronteiras entre regiões do mesmo nível no mapa, para cada nível. Recorre-se a um gerador do tipo *ConfigurableGenerator*;
6 Gera-se uma sequência possível de regiões marítimas nos diferentes níveis, recorrendo a um gerador do tipo *ConfigurableGenerator*. Não são efectuadas

permutações das regiões de um mesmo nível para se evitar situações de isomorfismo. Para além disso, todas as regiões terrestres que sejam fronteiras de pelo menos uma região marítima são convertidas em regiões costeiras para que o mapa permaneça consistente;

7 O mapa criado é inspeccionado para garantir a validade do mesmo. Em caso afirmativo, o mapa é submetido ao algoritmo de planeamento (capítulo 3) que o guarda na base de dados, com a respectiva solução. Em caso negativo o mapa é ignorado;

Este processo prossegue de forma iterativa sendo que:

- Para cada sequência de regiões do passo 1 consideram-se todas as sequências do passo 3;
- Para cada sequência de unidades do passo 3 consideram-se todas as sequências do passo 4;
- Para cada sequência de fronteiras do passo 4 consideram-se todas as sequências do passo 5.

Em cada iteração do passo 5 é obtida uma situação de jogo que, se for válida, é analisada pelo algoritmo *Min-Max* e guardada na base de dados. Desta forma, a base de dados é povoada com um conjunto de situações de jogo possíveis.

6

Sistema de Teste

Durante este trabalho foram também desenvolvidos e executados alguns mecanismos de teste ao software tanto do sistema de planeamento como do sistema de enumeração. Neste capítulo pretende-se analisar esses mecanismos e ferramentas utilizadas.

6.1 Testes Automatizados

Apesar da discussão e desenvolvimento de ferramentas para elaboração de testes automatizados ser algo relativamente recente, com os primeiros estudos desta área a aparecerem por volta de finais dos anos 90 [27] e com o grande desenvolvimento de plataformas de automatização de testes a dar-se já nos anos mais recentes, o princípio de construção deste tipo de testes tem vindo rapidamente a ganhar forma e importância na construção de software. A grande razão para tal crescimento prende-se com o facto de a crescente capacidade computacional dos sistemas computorizados permitir a criação de software cada vez mais poderoso e complexo. Com o aumento de complexidade surge também o aumento da probabilidade de erro humano no desenvolvimento de software. Assim surge cada vez mais a necessidade de testar o software, durante o seu desenvolvimento, de forma exaustiva e repetitiva. Apesar de tal poder ser feito de forma manual, constitui uma tarefa bastante trabalhosa e lenta pelo que, recorrendo a mecanismos de automatização de testes torna-se possível a execução de grandes quantidades de testes de forma repetitiva permitindo não só a detecção e correcção de erros existentes, como também um certo nível de protecção contra a criação de erros, em zonas previamente estáveis, resultantes de expansões ou actualizações ao software.

6.1.1 JUnit & JMock:

Actualmente, o *JUnit* e *JMock* apresentam-se como duas boas ferramentas de auxílio à criação de testes automatizados na linguagem *Java*, sendo o *JMock* um complemento do *JUnit*.

O *JUnit* trata-se de uma instância para *Java* da arquitectura de criação de testes automatizados *XUnit* que permite testar, de forma isolada, os métodos das classes que constituem um programa em *Java*. Desta forma é possível criar um conjunto de classes com métodos de teste que invocam um determinado método de uma classe, com vários valores de entrada possíveis de forma a analisar os resultados obtidos com os resultados esperados dessas invocações bem como tempos de execução dessas invocações.

Neste trabalho foram elaborados testes automatizados que, de forma rápida, apresentam os resultados de invocações a métodos de contagem de instruções, unidades e regiões das classes relativas às estruturas de representação de mapas e instruções. Foram também desenvolvidos testes aos tipos enumerados por forma garantir o seu correcto funcionamento em termos de conversão para formato textual para escrita nos ficheiros da base de dados e na conversão de volta para o seu valor em termos de código. Foram ainda implementados testes ao comparador da base de dados de forma a testar a sua capacidade de detectar equivalência entre mapas referida do capítulo 4.

Por fim, foram elaborados testes ao planeador para avaliar a sua capacidade de detectar conjuntos de instruções inválidas decorrentes do funcionamento do algoritmo *Min-Max* estudado no capítulo 3. No entanto, nesta fase surgiu a necessidade de se testar a classe *Plan* de forma isolada. Tal era impossível apenas com *JUnit* devido à dependência que esta classe apresenta em relação à classe que implementa a interface *Leadership*. Esta dependência forçava a existência de um objecto da classe *Commander* para ser fornecido ao objecto da classe *Plan*. A solução para esta questão encontra-se na ferramenta *JMock* pois a mesma permite a substituição, para efeitos de execução de testes, dos objectos necessários, por objectos falsos denominados "*Dummys*", cujo comportamento é definido antes da execução dos testes. A única condição imposta é o facto da classe do objecto que se pretende substituir ter que implementar uma interface, pois é através da sua interface que o *JMock* obtém os métodos públicos com os quais se define o comportamento do *dummy*. Desta forma foi possível criar um *dummy* da interface *Leadership* e definir o seu comportamento em cada teste executado de forma a que este objecto falso retornasse os valores pretendidos em cada situação.

A criação de um *dummy* consiste apenas na definição da interface que o mesmo representa, seguida dos métodos que irão ser invocados pela respectiva ordem e consequentes respostas. Esta técnica não só permite isolar os componentes como também permite

efectuar testes ao nível da interacção entre os vários componentes, alertando o programador nas três situações seguintes:

- Uma invocação esperada não ocorreu durante o teste;
- Ocorreu uma invocação após já terem ocorrido todas as invocações esperadas;
- Ocorreu uma invocação quando era esperada uma outra.

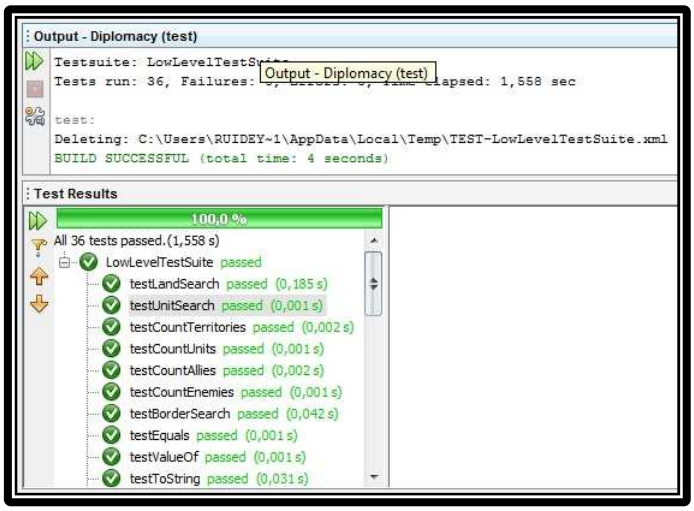


Figura 6.1 - Resultados de testes unitários

Devido à integração das ferramentas *JUnit* e *JMock* em *IDEs* para programação em *Java*, é possível apresentar os resultados dos testes executados numa forma visual que permite ao programador perceber rapidamente quais os testes que estão a falhar e porquê, qual a percentagem de testes com e sem sucesso e tempos de execução de testes em microssegundos com se pode ver na figura 6.1.

6.2 Testes Manuais

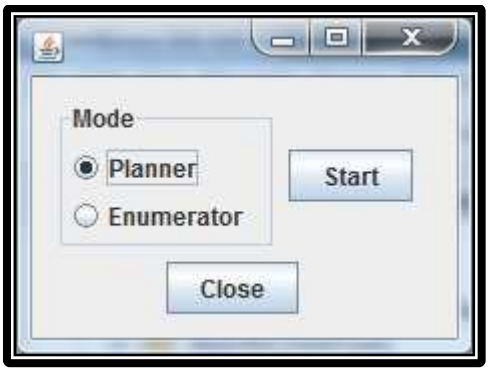


Figura 6.2 - Menu Principal

Apesar de todo o trabalho realizado ao nível de automatização de testes neste trabalho, foi desenvolvido também uma ferramenta gráfica para execução de testes manuais ao software. Essa ferramenta é constituída por um conjunto de *GUIs*, criados com recurso aos componentes *Swing* do *Java*, que permitem a criação de mapas para submissão ao sistema de planeamento, bem como a obtenção de mapas do sistema de enumeração estudados anteriormente. Esses mapas são representados graficamente recorrendo a um software descrito na secção 6.2.1. Tais componentes gráficos são facilmente acedidos pelo menu incluído, com mostra a figura 6.2.

6.2.1 O Software GraphViz:

Para poder apresentar um mapa no monitor num formato gráfico era necessário encontrar uma forma de representar graficamente um grafo. Para tal recorreu-se a um software externo denominado *GraphViz* [28] que permite a criação de um ficheiro em formato de imagem (jpg, bmp, gif, etc.) com a representação gráfica de um grafo a partir de um ficheiro com a definição das características e elementos do grafo na linguagem de programação *Dot*. Apesar de o *GraphViz* também fornecer ferramentas para a criação do grafo na linguagem *Dot*, recorreu-se a uma classe criada, em *Java*, por ***Laszlo Szathmary*** **[29] e disponibilizada na internet pelo mesmo, que fornece um meio fácil de interacção com o software *GraphViz*, desde que o mesmo esteja correctamente instalado no computador. Com recurso a estes elementos foi possível implementar um mecanismo representação gráfica de mapas eficiente e fácil de utilizar.**

Note que a ferramenta gráfica tem um mecanismo de validação de mapas mínimo. Assume-se que o utilizador não cria mapas incoerentes, como por exemplo mapas representados por grafos não conexos, pelo que os mecanismos de validação e detecção de erros incluídos nestas interfaces são mínimos.

6.2.2 Interface de Planeamento:

A interface de interacção com o sistema de planeamento é constituída por uma interface de criação do mapa e inserção de unidades e por uma interface de amostragem da solução obtida, ver figuras 6.3 e 6.4.

A interface de criação de mapas contém ferramentas simples para a criação de regiões e suas fronteiras, bem como o posicionamento de unidades nas regiões criadas. Após esse processo, é possível inserir o número de turnos e pedir a solução ao software que a mostrará na interface de amostragem constituída por um painel com a representação do mapa criado juntamente com uma lista onde são escritas as instruções referentes ao turno escolhido na caixa de selecção providenciada.

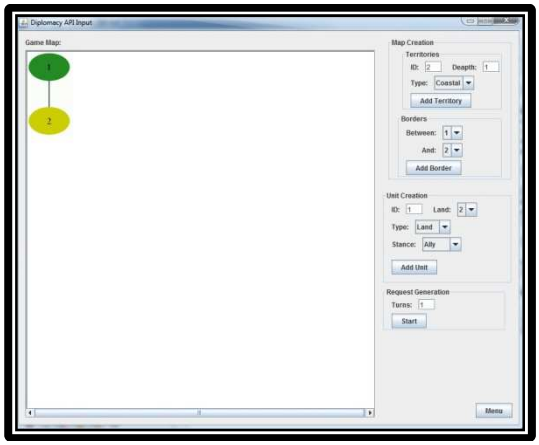


Figura 6.3 - Interface de Entrada do Planeador

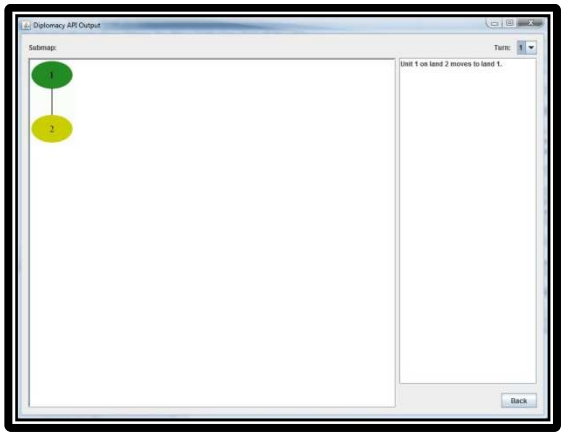


Figura 6.4 - Interface de Saída do Planeador

6.2.3 Interface de Enumeração:

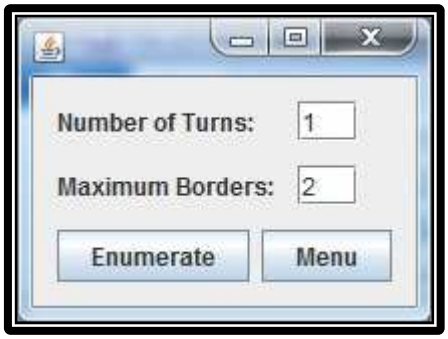


Figura 6.5 - Menu do Enumerador

Por sua vez, a interface de interacção com o sistema de enumeração é constituída por uma interface de inserção de parâmetros e por uma interface de amostragem de mapas, ver figuras 6.5 e 6.6.

A interface de inserção de parâmetros é constituída por campos que permitem a inserção do número de turnos e do número máximo de fronteiras de cada região que são fornecidos ao sistema de enumeração. Após o processo de enumeração, que pode demorar vários minutos para valores relativamente baixos, é apresentada a interface de amostragem que é muito semelhante à interface de amostragem do sistema de planeamento com a diferença de que a lista presente no lado direito da interface mostra agora as unidades existentes no mapa. Além disso, no canto inferior direito existe um botão *Next* que permite a visualização do próximo mapa enumerado.

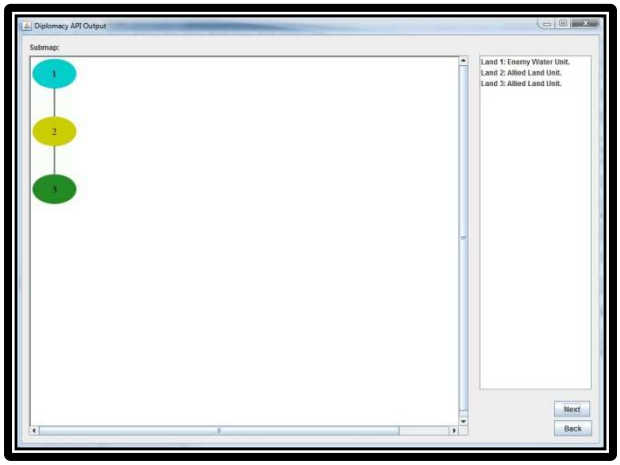


Figura 6.6 - Interface de Saída do Enumerador

7

Conclusão

Nesta tese é apresentada uma possível abordagem ao problema de planeamento e decisão em *Jogos de Soma-Zero* de 2 Jogadores com utilização de base de dados e do princípio de *Min-Max* no contexto do jogo *Diplomacy* (capítulos 3 e 4). No entanto, apesar de se ter cumprido este objectivo, o algoritmo de planeamento, devido à sua natureza recursiva e combinatória, revela as suas limitações relacionadas com o esforço computacional necessário para resolver situações de jogo com uma dimensão relativamente pequena. Apesar de se poder introduzir mecanismos de concorrência ou melhoramento da pesquisa recursiva subjacente ao princípio *Min-Max*, sua eficiência dependerá muito do hardware existente no sistema computacional.

Para cumprimento dos objectivos propostos, é também apresentada uma estratégia de enumeração de situações de jogo para inicialização da base de dados (capítulo 5). Os princípios de enumeração de grafos auxiliam a implementação deste tipo de mecanismos. No entanto, as questões inerentes ao problema de isomorfismo de grafos condicionam a eficiência e abrangência das enumerações efectuadas. O elevado esforço computacional torna-se menos relevante devido a este sistema apenas servir para efeitos de inicialização da base de dados, significando que, em situação normal, será executado apenas na primeira utilização do software.

Os principais contributos deste trabalho são:

1. A implementação de um mecanismo de planeamento de jogadas com recurso ao princípio Min-Max, e uma breve análise da complexidade desta implementação;
2. A implementação de um mecanismo de base de dados sob a forma de armazenamento em ficheiros;
3. A definição e implementação de uma estratégia de enumeração de situações de jogo com recurso a princípios de enumeração de grafos e isomorfismo;

4. A implementação de testes automatizados a alguns componentes do software com recursos às ferramentas JUnit e JMock existentes na linguagem de programação Java (capítulo 6);
5. A implementação de uma interface gráfica para efeitos de execução de testes manuais ao software através da ferramenta Swing da linguagem Java;
6. A definição e implementação de uma API de forma a disponibilizar o mecanismo de planeamento para uso por parte de uma entidade exterior ao software.

8

Trabalho Futuro

A conclusão deste trabalho permite criar perspectivas no sentido da integração deste software numa possível implementação de um agente de inteligência artificial capaz de jogar o jogo *Diplomacy*, podendo esse agente ser criado no âmbito do projecto *DAIDE* ou de forma independente. Poderá também ser considerada a possibilidade de aperfeiçoar a arquitectura da base de dados com recurso a ficheiros *XML* ou outras arquitecturas de armazenamento de informação.

É ainda deixada em aberto a hipótese de se efectuar um estudo ao nível da área de Teoria de Jogos no sentido de se analisar o problema de planeamento em Jogos de Soma-Zero de N Jogadores de forma a eliminar a necessidade de divisão dos jogadores em dois grupos, aliados e inimigos, assim como para introduzir a questão dos jogadores neutros no algoritmo de planeamento. Por outro lado, a consideração de N Jogadores poderá suscitar a necessidade de implementar um software capaz de identificar as relações entre os vários jogadores. Esta questão criará também a possibilidade de modificar o algoritmo de planeamento e a base de dados para se poder considerar várias divisões possíveis dos N Jogadores em grupos. O planeamento baseado nesses grupos poderá fornecer informações sobre as várias alianças possíveis, e as mais vantajosas.

Por último, poderá ser considerada a possibilidade de alteração do software com vista à inclusão de mecanismos que permitam efectuar o planeamento de várias situações de jogo de forma concorrente.

Anexo A – Diagramas

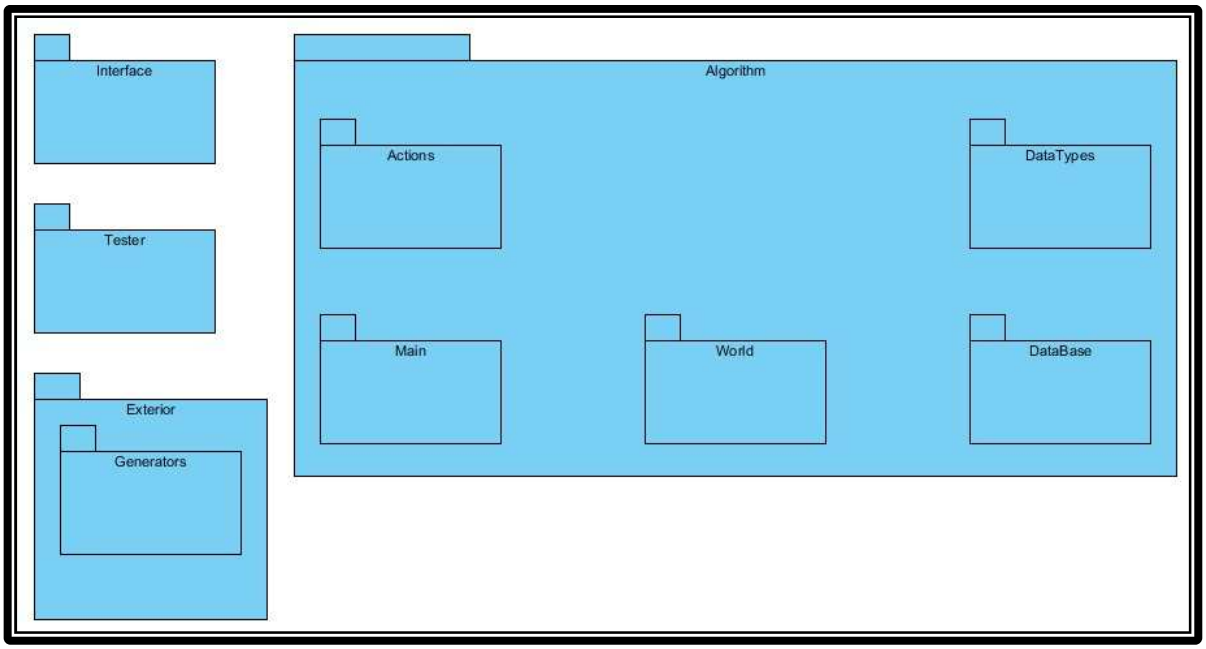


Diagrama 1 - Diagrama com a estrutura de pacotes do software

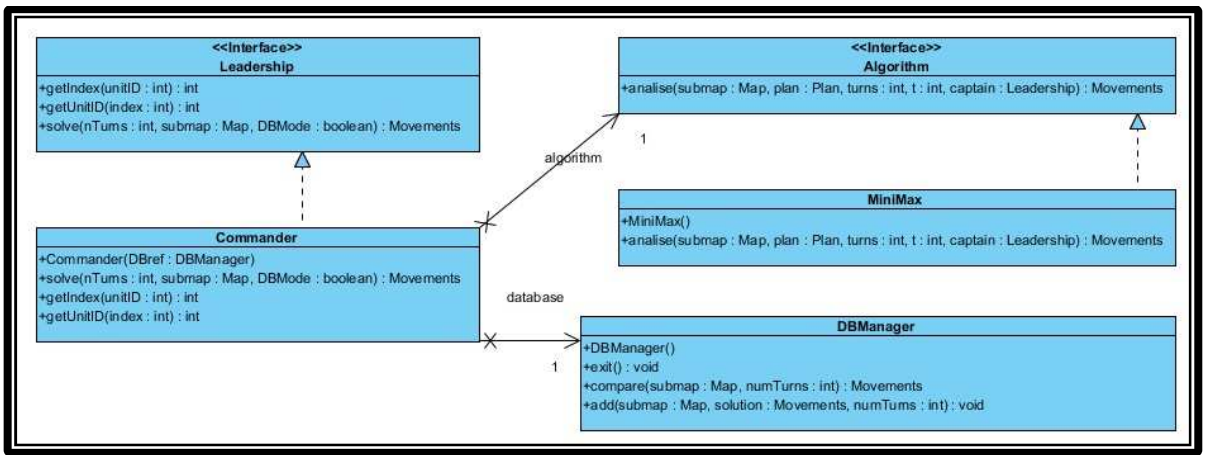


Diagrama 2 - Diagrama com as entidades de planeamento e base de dados

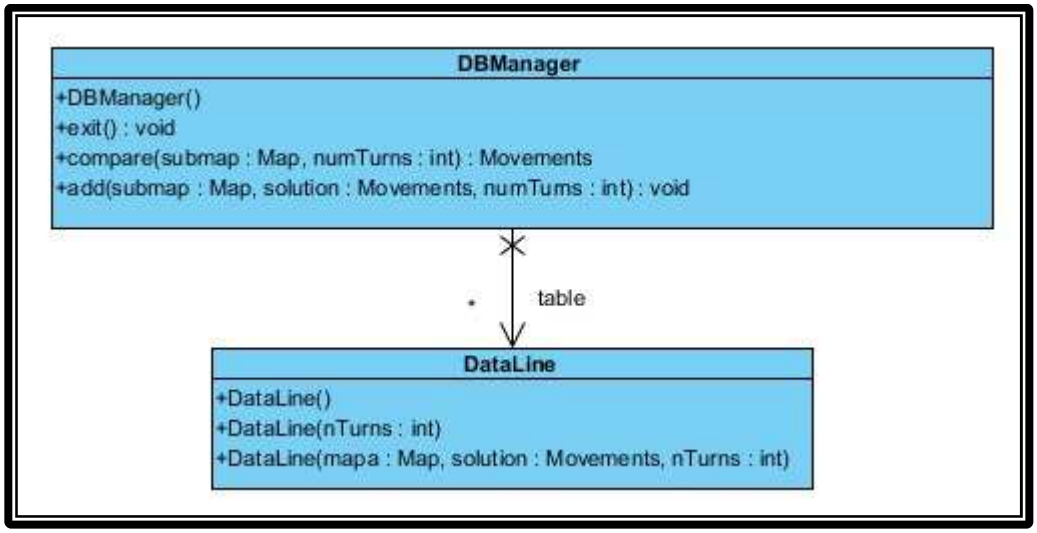


Diagrama 3 - Diagrama da estrutura da base de dados

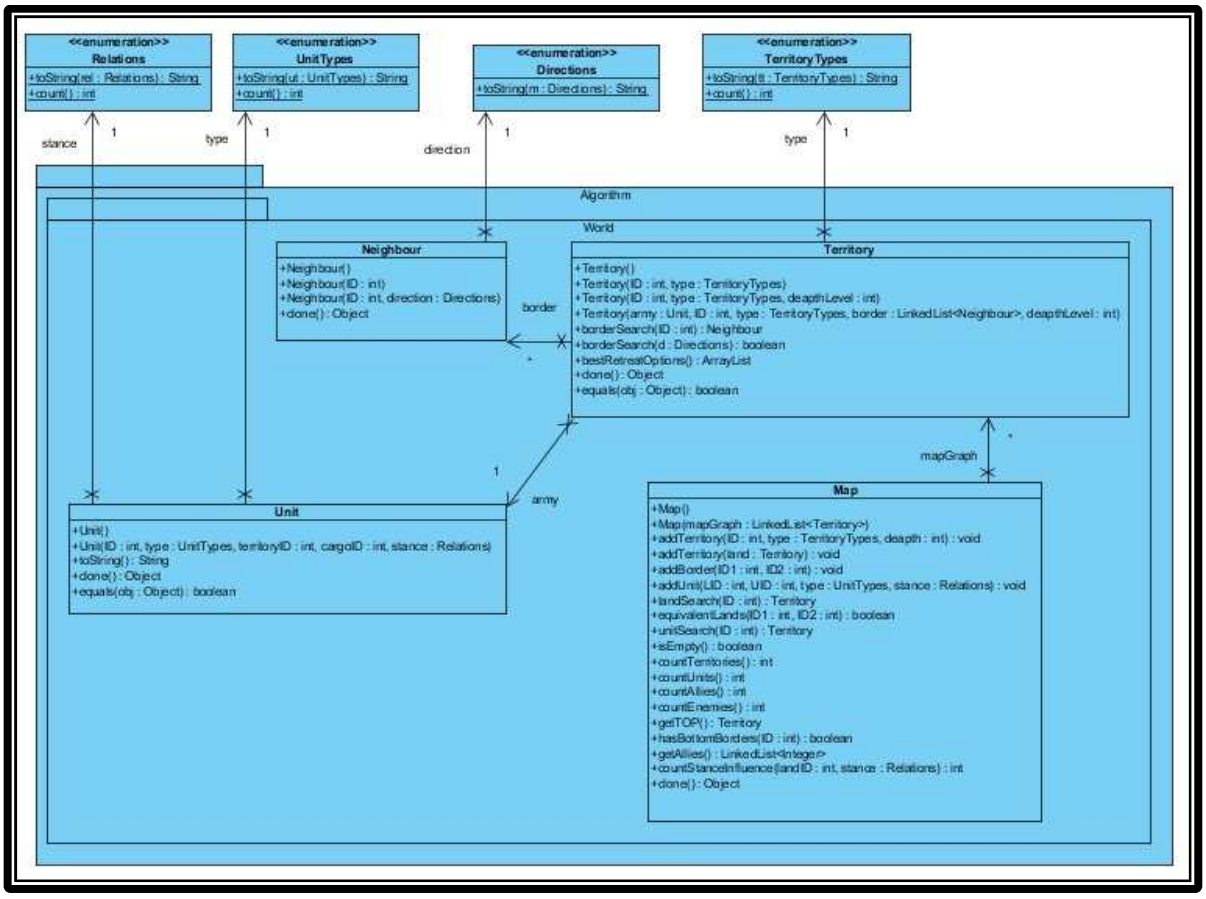


Diagrama 4 - Diagrama de classes da estrutura de dados para os mapas

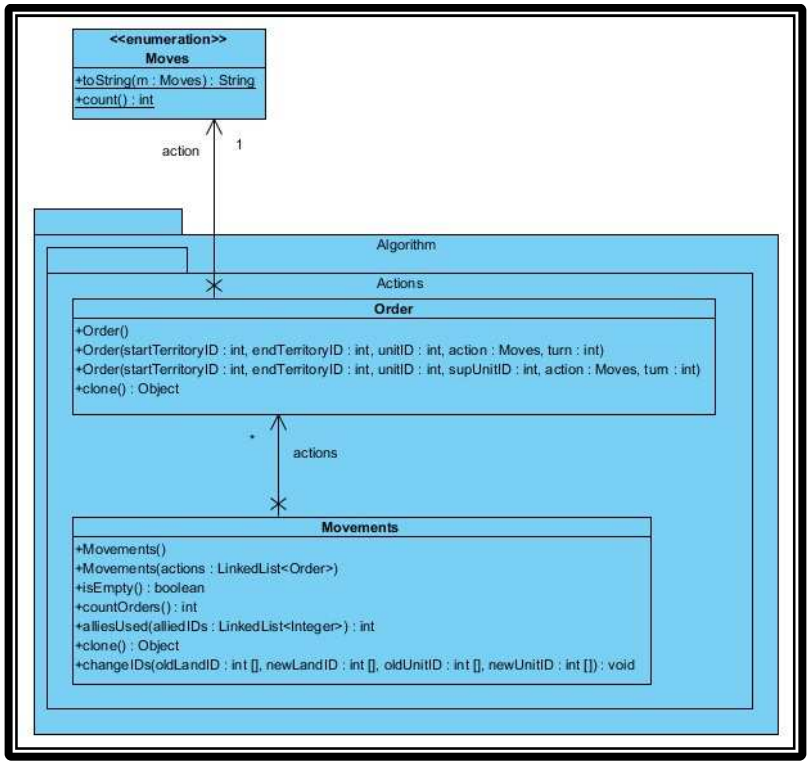


Diagrama 5 - Diagrama de classes da estrutura de dados das soluções

Bibliografia

[1]. Wikipedia, the free encyclopedia. Diplomacy (game). *Board Games*, 2010. http://en.wikipedia.org/wiki/Diplomacy_(game)

[2]. webDiplomacy. *A multiplayer web implementation of the popular turn-based strategy game Diplomacy*. http://www.webdiplomacy.net/

[3]. Paradox Interactive Corporate Web Page. http://www.paradoxplaza.com/corporate

[4]. Wizards of the Coast LLC. Avalon Hill Official Home Page, 1995. http://www.wizards.com/default.asp?x=ah/welcome

[5]. IBM Research Deep Blue Overview, 1997. http://www.research.ibm.com/deepblue/

[6]. Andrew Rose & Hamish Williams. Diplomacy AI Centre, 2003. http://www.daide.org.uk/index.xml

[7]. David. David's Diplomacy AI Page. http://www.ellought.demon.co.uk/dipai/

[8]. John Newbury. *AI in the Game of Diplomacy: the BlabBot Bot*, 2008. http://johnnewbury.co.cc/diplomacy/blabbot/index.htm

[9]. Jason van Hal. Diplomacy AI – Albert. http://sites.google.com/site/diplomacyai/

[10]. Michael Jones. Michael Jones's Brutus – DAIDE Project, 2010. http://sites.google.com/site/daideproject/

[11]. John L Casti. *Five golden rules: great theories of 20th-century mathematics – and why they matter*. New York: Wiley-Interscience. p. 19, 1996.

[12]. Wikipedia, the free encyclopedia. Minimax. *Game Theory*, 2010. http://en.wikipedia.org/wiki/Minimax

[13]. Russell, Stuart J.; Norvig, Peter, *Artificial Intelligence: A Modern Approach* (2nd edition.), Upper Saddle River, New Jersey: Prentice Hall, pp. 163–171, 2003.

[14]. Wikipedia, the free encyclopedia. Game theory. *Artificial intelligence*, 2010. http://en.wikipedia.org/wiki/Game_theory

[15]. Dutta, Prajit K. *Strategies and games: theory and practice*, MIT Press, 1999.

[16]. Gintis, Herbert. *Game theory evolving: a problem-centered introduction to modeling strategic behavior*, Princeton University Press, 2000.

[17]. Leyton-Brown, Kevin; Shoham, Yoav. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*, 2008.

[18]. Rasmusen, Eric. *Games and Information: An Introduction to Game Theory* (4th edition), 2006.

[19]. Wikipedia, the free encyclopedia. Graph Enumeration. *Combinatorics,* 2010 http://en.wikipedia.org/wiki/Graph_enumeration

[20]. Harary, Frank; Schwenk, Allen J., "The number of caterpillars", *Discrete Mathematics* 6 (4): 359–365, 1973.

[21]. Wikipedia, the free encyclopedia. George Pólya. http://en.wikipedia.org/wiki/Polya

[22]. Wikipedia, the free encyclopedia. Arthur Cayley. http://en.wikipedia.org/wiki/Arthur_Cayley

[23]. Wikipedia, the free encyclopedia. John Howard Redfield. http://en.wikipedia.org/wiki/John_Howard_Redfield

[24]. H. Whitney, "Congruent graphs and the connectivity of graphs", *Am. J. Math.*, 54 pp. 160-168, 1932.

[25]. Wikipedia, the free encyclopedia. Graph isomorphism. *Graph Theory*, 2010. http://en.wikipedia.org/wiki/Graph_isomorphism

[26]. Dirk L. Vertigan, Geoffrey P. Whittle: A 2-Isomorphism Theorem for Hypergraphs. *Combinatorial Theory*, Ser. B 71(2): 215-230, 1997.

[27]. Elfriede Dustin, *Automated Software Testing.* Addison Wesley, 1999

[28]. AT&T. Graphviz – Graph Visualization Software. http://www.graphviz.org/

[29]. **Laszlo Szathmary**. GraphViz Java API Version 0.2, 2003 http://webloria.loria.fr/~szathmar/off/projects/java/GraphVizAPI/index.php